ELSEVIER
Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®
Computer Networks xxx (2003) xxx–xxx
COMPUTER NETWORKS
www.elsevier.com/locate/comnet

Overload control in QoS-aware web servers ☆

Huamin Chen, Prasant Mohapatra *

Department of Computer Science, University of California, One Shields Avenue, Davis, CA 95616, USA

Received 14 November 2002; accepted 16 January 2003

Abstract

With the explosive use of Internet, contemporary web servers are susceptible to overloads during which their services deteriorate drastically and often lead to denial of services. Overloads are of more serious concerns for QoS-aware servers. Evaluation of performance of QoS-aware servers in terms of the number of request completion is not very meaningful. A better measure would be the number of completed sessions. In this paper, we proposed two methods to prevent and control overloads in web servers by utilizing session-based relationship among HTTP requests. We first exploited the dependence among session-based requests by analyzing and predicting the reference patterns. Using the dependency relationships, we have derived traffic conformation functions that can be used for capacity planning and overload prevention in web servers. Second, we have proposed a dynamic weighted fair sharing (DWFS) scheduling algorithm to control overloads in web servers. DWFS is distinguished from other scheduling algorithms in the sense that it aims to avoid processing of requests that belong to sessions that are likely to be aborted in the near future. The experimental results demonstrate that DWFS can improve server responsiveness by as high as 50% while providing QoS support through service differentiation for a class of application environment.

Keywords: Capacity planning; Dynamic weighted fair sharing; Overload control; Quality of service; Scheduling algorithm; Service differentiation; Session-based control; Web server

1. Introduction

As the widespread usage of web service grows, the number of accesses to many popular web sites is ever increasing and occasionally reaches the limit of their capacity and consequently causes the servers to be overloaded. As a result, end users either receive busy signal or nothing at all before the browser indicates a time-out error or the user aborts (stops) the request. Subsequently, the server may get choked or crash causing denial of services. Such abnormality is often regarded as the servers' poor quality and compromises their long term survivability. In e-commerce applications, such server behavior could translate to sizable revenue losses.

Research on overload prevention and control has been limited compared to the other performance improvement issues such as web caching, and load balancing in web servers. These perfor-

☆ A preliminary version of this paper appeared in the Proceedings of IEEE INFOCOM 2002. This research was supported in part by the National Science Foundation through the grants CCR-0296070 and ANI-0296034.

* Corresponding author.
E-mail addresses: chenhua@cs.ucdavis.edu (H. Chen), prasant@cs.ucdavis.edu (P. Mohapatra).

1389-1286/02/$ - see front matter © 2002 Published by Elsevier Science B.V.
doi:10.1016/S1389-1286(03)00178-6

mance enhancement techniques, however, are inadequate in ensuring a busy web server from being overloaded due to the fact that the web traffic is highly unpredictable and bursty [10,15]. Proper capacity planning and forecasting methods can prevent servers from being overloaded under controlled traffic conditions.

In many web sites, especially in e-commerce, online brokers, and supply chain sites, majority of the requests in the web traffic are session-based. A session contains temporally and logically related request sequences from the same client. Sessions can be identified either by HTTP/1.1 persistent connections [12] or from the state information within the presence of cookies [14]. Sessions exhibit distinguishable features from individual requests. For example, session integrity requires that once admitted for processing, all the following requests within a session should be honored. Similarly, session affinity would require that requests belonging to the same session are handled by the same front-end server for security and locality reasons. These features may complicate or contradict the research conclusions of the performance studies on web servers where the number of request completions have been considered as the primary performance measure. For example, admission control on a per request basis may lead to a large number of broken or incomplete sessions when the system is overloaded. Incomplete sessions may be equivalent to a rejected session from the users viewpoint or for most e-commerce servers. Thus, performance measure based on the number of request completions may not be a good indication of users satisfaction (the basic purpose of web service). Especially during overloads, the disparity between the two types of performance measures (proportion of request completion and proportion of session completion) is more enhanced. Capacity planning schemes based on individual requests also have the same deficiency.

Session integrity is a critical metric for commercial web service. For an online retailer, the more the number of sessions completed, the more the amount of revenue that is likely to be generated. The same statement cannot be made about the individual request completions. Sessions that are broken or delayed at some critical stages, like checkout and shipping, could mean loss of revenue to the web site. From the end users' perspective, this means poor service availability. Therefore, it is more useful to use session integrity to evaluate the service availability of servers, especially during high-load periods.

In this paper, we explore the session characteristics and their potential in overload control and prevention. A workload characterization study is done first to gain an insight to the load patterns in web servers. The workload characterization study was based on the server log from a popular online retailer. We found that, despite the seemingly complication of session sequences, some statistical results can used in simplifying the session-based traffic model. Based on these results, the session logic can be utilized for capacity planning and request scheduling of QoS-aware servers, which improves server's productivity. Server productivity quantifies the amount of useful work done by the server. Based on the session-level traffic model, we have proposed a dynamic weighted fair scheduling (DWFS) scheme that assign service weight to different requests of a session in a dynamic manner. We have done an experimental performance analysis by modifying the scheduling scheme of the Apache web server. The proposed DWFS scheme provides a performance improvement of about 50% in terms of response delay and significantly reduces the session abortion rate for the workload and system configuration used in the experimentation.

The rest of the paper is organized in the following manner. Section 2 characterizes session-based HTTP requests. Section 3 provides capacity planning tools to prevent server overload. Section 4 proposes a request scheduling algorithm to control server overload and improve server performance followed by experimental results in Section 5, which proves the feasibility and quantifies the performance of the proposed algorithm. The related works are discussed in Section 5, followed by the concluding remarks in Section 7.

2. Session-based web traffic characterization

A session in web accesses can be defined as a sequence of requests that form one complete

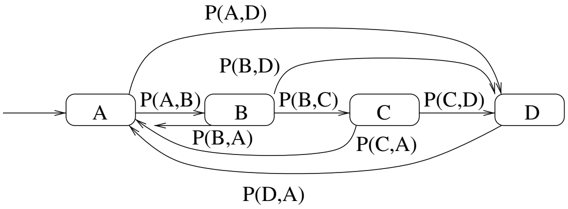

Fig. 1. An example of a web session represented as a state machine.

transaction. A session during web accesses can be represented as a finite state machine with each state representing a stage that a request is undergoing. Fig. 1 depicts an example of such a representation. The directed arc (A,B) represents a transition from state A to state B with a probability $P(\mathrm{A}, \mathrm{B})$. The four states A, B, C, and D could be representing states corresponding to main menu, checkout, browsing and search in an e-commerce site.

In web services, a stage can be a single URL or a group of URL's that have the same reference pattern and resource claim profile. A session can be mandatory or voluntary. A mandatory session refers to the situation in which the descendant requests are generated by the browsers instead of clients. The requests for embedded image files within an HTML page is an example of this case. We call the page that has embedded files as the *main page*. In voluntary sessions, the descendant requests are generated by the clients explicitly. For example, the client clicks a link within the current main page to browse another page. In current web server architecture, most of the image files are served by edge servers [2] or dedicated image servers which are physically separated from those serving the main pages. Therefore, the performance of these servers is largely affected by the service of main pages. Thus, the following discussion is focused on voluntary sessions.

During our research, we obtained the trace of accesses to an e-commerce web server from a popular online retailer. [1] Based on the reference traces, we characterized the basic behavior of session-based traffic. The characterizations are derived from a typical daily traffic trace. Previous studies [10] have characterized the web traffic as very bursty, which is also observed in our result as shown in Fig. 2. From Fig. 2, it is observed that the traffic load is highest during the period of 17:00–23:00 h, which accounts for over 50% of the daily traffic. The traffic volume peaked at 20:00–21:00, where nearly 10% of overall requests were initiated. So the server is presumably more heavily loaded during this period (during evening hours), which was confirmed from the server side performance data recorded by the MS Performance Monitor.

We further investigate the relationship between request queue length (the waiting requests and those being served) and the request processing time under heavy server load. Since the processing time for individual URL's varies, we adopt the measure called *stretch factor* from [23], which refers to the quotient between the current processing time and the processing time of the same URL under normal load. The stretch factor reflects the current server load. Fig. 3 depicts the queue length and the stretch factor during the 20:00–21:00 period. It is observed that the two curves show similar pattern, indicating that the server load is proportional to the queue length. While the queue length is a good indication of server load, the nature of the jobs in the queue also has a great impact on server performance. Servers whose

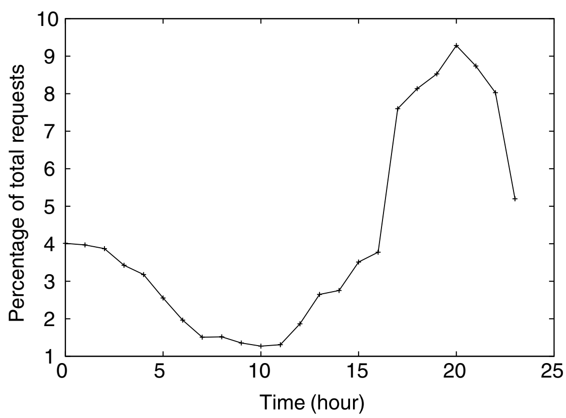

Fig. 2. Traffic histogram of the server trace for a day.

[1] We refrain from mentioning the name of the retailer honoring a non-disclosure agreement. Without the non-disclosure agreement, we would have been able to obtain the data.

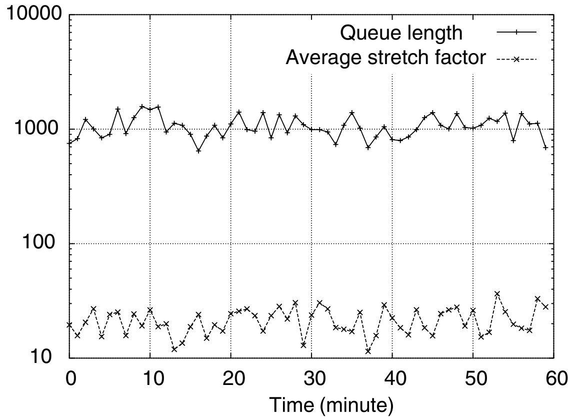

Fig. 3. Queue length and stretch factor.

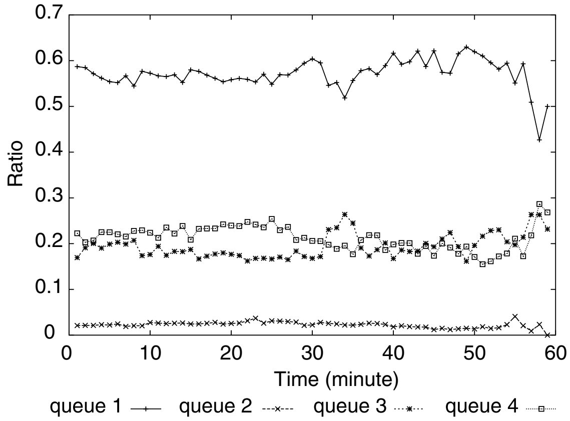

Fig. 4. Queue composition for the 1 h duration.

workload is dominated by static files (like HTML pages, images) generally perform better than the ones with high proportion of dynamic files (like CGI, ASP, etc). Therefore, unless the composition of workload is comparable, using the queue length as an indicator of server load is incorrect.

We sorted the requests into different queues according to their nature (main menu, checkout, search, browsing) and analyzed the composition of workload at each time period. These queues usually have different resource consumption profiles. For main menu queues, the major task is static HTML and image files rendering, and the update frequency is relatively low. For checkout queues, the process is SSL-secured and thus the workload is very CPU intensive. For search queues, the back-end database is queried and the server only receives the query results and assembles them in the HTML format. Characteristics of the browsing queue is like main menu queue except that more image files are rendered. Fig. 4 presents the nature of the workload composition during the period 20:00–21:00, which is observed to be relatively stable.

It has been realized in prior studies that the requests within a session reveal statistically dependent relationship [7,17]. Conclusions from these studies show that historic reference patterns can be exploited to predict the subsequent requests. Prediction method of subsequent requests within a session are different. In this study, we use the transition probability of the state machine for prediction, which derives the subsequent URL from the current one. This method requires no sophisticated mathematical modeling and uses less computation power in practice. The probability is obtained either from offline historic records like server logs or from online statistics. The work in this paper is based on offline record of server logs, however other methods can also be applied in a similar way. We will later show that even with this simple prediction method, the performance improvement is significant.

Fig. 5 shows the transition probability (in percentage) matrix among the stage vector components: main menu (MM), checkout (CO), search (SR), browsing (BR) from the online retailer's server log. The row vectors show the transition from one stage to another. For example, the second row indicates that the transition possibility from CO to MM, CO, SR, and BR is 13.5%, 44.9%, 40.2%, and 1.4%, respectively. The server log file format follows W3C extended logging [18]. Each request entry contains a user ID for the login user which facilitates the identification of session owners. Session integrity is maintained by the IIS server (Microsoft Internet Information Server).

	MM	CO	SR	BR
MM	74.8	0.6	19.5	5.1
CO	13.5	44.9	40.2	1.4
SR	18.1	3.4	74.3	4.3
BR	13.7	1.0	14.6	70.7

Fig. 5. State transition matrix.

Another issue related to the session behavior is the user thinking time between consecutive requests within a session. It is random in nature and varies for different web sites. In our server trace, the thinking time was usually short and less than 60 s. Characterization of other traces reveals similar results. When the traffic is high, which is the case for heavily loaded servers, the long term effect of the thinking time can be ignored.

Finally, when evaluating the relationship between the number of outstanding requests and the number of active sessions, we found that the ratio between the two is stable. Though each session can fork several requests simultaneously, the fact that some others do not send any requests offsets it, which makes the overall behavior as stable. Fig. 6 shows this ratio for the trace of the period 20:00–21:00. In this figure, the average value is 0.526 with a standard deviation of 0.017, which means that the variation is small and the ratio is stable. This result is useful in estimation of the request arrival rate based on the number of active sessions.

Obtaining server traces from e-commerce sites has been difficult due to security and proprietary reasons. We could manage to get the traces from one corporation. Although our analysis and results use only this set of trace, the proposed methodology is applicable for any other server trace. So we have laid emphasis on the methodology, the trends, and relativeness of the results, rather than the absolute numbers (which are specific for the trace).

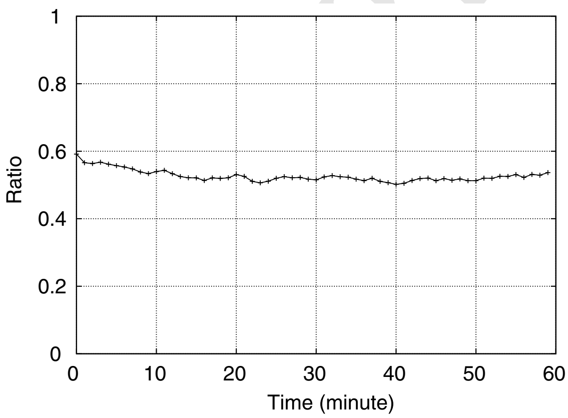

Fig. 6. Ratio of request arrival rate to session generation rate.

3. Session-based capacity planning

We have considered QoS-aware web servers for our study in which requests are served based on their priority levels. The basis of priority assignment is actively discussed in [11] and is beyond the scope of this study. In QoS-aware web servers, an important and interesting performance metric is the delay bound, which is the maximum response delay a request encounters. Besides the processing time of each request, there are other latencies associated with the service of a request, such as queuing delay and network transmission delay. In this paper, we only consider the delay at the server side. The study of network delay is not within the scope of this paper and is being extensively studied in the IETF architectures [11]. We assume that the service level agreement (SLA) that can be provided by a web server would consist of a bounded delay for each QoS level if the request arrival rate does not exceed an agreed amount. From the web server perspective, QoS attributes are defined in terms of the maximum rate of request arrival and the latency bound of each request. An SLA can thus be stated in terms of (λ, δ, s), where λ is the maximum rate of arrival, δ is the delay bound, and s is the proportion of requests that meet the delay bound. In a QoS-aware web server, overload is said to occur when the SLA is violated for an extended period of time. Thus, we formally define overload as follows.

> Let the predefined SLA for a specific QoS level is (λ, δ, s). If for an extended period of time T, while the arrival rate is less than λ, the proportion of requests that meet the delay bound stay less than s, the web server is said to be overloaded. It is implicit in this definition that the SLA negotiation was done considering the server capacity and workload conditions.

The following analysis focuses on the worst case where requests compete for CPU resources (Table 1). The QoS rules are defined for each of the URL's or for URL groups. The SLA specifies the delay bound of the QoS groups when the request arrival rate is below some threshold.

Table 1
Notations used for the analysis

Notations	Description
$P_{a,b}$	State transition probability from a to b
λ_a^i	Request arrival rate of session class i to state a
μ_a^i	Request departure rate of session class i from state a
d_a^i	Delay bound of class i at stage a
T_a	Processing time at stage a
ϕ_i	Session generation rate of class i
r_{rs}	Ratio between the number of requests and the number of sessions

We only consider a steady-state system because a transient model is mathematically intractable and may be of little practical use. In a steady-state system, the number of requests arriving at and departing from the server must be equal. Thus, the arrival rate λ_i^p is equal to the departure rate μ_i^p under the steady-state assumption.

The turnaround time at each stage is less than the lower limit, which is the sum of the delay bound and the processing time. Thus,

$$\mu_i^p \geqslant \frac{1}{T_i + d_i^p}.$$

The input to each stage is the output from the other stages with certain transition probability. Excluding the source and sink stages, we then have

$$\lambda_j^p = \sum \mu_i^p * P_{i,j} = \mu^p \cdot P_j,$$

where μ^p and P_j are the vector forms of μ_i^p and $P_{i,j}$ and the second expression is the dot product.

With a pre-emptive priority scheduling discipline, the high priority requests are scheduled before the low priority ones. Thus the departure rate of a priority group is less than the service rate of higher priority groups.

$$\frac{1}{T_i + d_i^p} \leqslant \mu_i^p \leqslant \frac{\lambda_i^p}{\sum_{j=1}^{p} \lambda_i^j * T_i}. \tag{1}$$

Using the steady-state assumption,

$$\frac{1}{T_i + d_i^p} \leqslant \lambda_i^p \leqslant \left[\frac{\lambda_k^p}{\sum_{j=1}^{p} \lambda_k^j * T_k} \right] \cdot P_i. \tag{2}$$

Finally, using (2) for all the stages and presenting in a matrix form, we get:

Eq. (3) reveals the relationship between delay and traffic volume of the QoS priority groups. To infer

$$\left[\frac{1}{T_i + d_i^p} \right]_{m*n} \leqslant [\lambda_i^p]_{m*n} \leqslant \left[\frac{\lambda_k^p}{\sum_{j=1}^{p} \lambda_k^j * T_k} \right]_{m*n} * [P]_{n*n}, \tag{3}$$

where m is the number of priority groups and n is the number of stages. We call this expression as the traffic conformation inequation

the SLA from this function, we can analyze in a stepwise manner. It is obvious that for the highest priority, the request arrival rate at stage i is constrained by

$$\frac{1}{T_i + d_i^1} \leqslant \lambda_i^1.$$

In practice, the λ_i^1 can be set to its lower bound so that they will not cause excessive delay for the lower priority groups and maximize system utilization by allowing more requests into the system. Thus the arrival rate of the next immediate priority group is bounded by

$$\frac{1}{T_i + d_i^2} \leqslant \lambda_i^2 \leqslant \left[\frac{\lambda_k^2}{\sum_{j=1}^{2} \lambda_k^j * T_k} \right] * P_i.$$

Similarly, the arrival rate of other priority groups can be obtained.

After obtaining the request arrival threshold, it is easy to define the session generation rate using the ratio r_{rs}. As discussed in Section 2, r_{rs} is more or less stable when the traffic volume is high. Thus the session generation rate of class i is given by

$$\phi_i = r_{\mathrm{rs}} \sum \lambda_j^i. \tag{4}$$

4. Productive scheduling algorithm

Among the several causes that are responsible for the degradation of server output, scheduling of requests is a critical factor. For example, queues consisting of time-consuming requests have a good chance of getting accumulated. As a result, they would dominate in controlling CPU resources in a round robin scheduling mode. Consequently, more time is spent on these queues and the effective overall output is degraded. Intuitively a conservative admission control can prevent the server from being overloaded. But such conservativeness is not easy to realize because of the burstiness of traffic and the likely-hood of leading to underutilization of the server. We believe a relatively relaxed admission control assisted by an efficient scheduling algorithm is a better alternative. The admission control admits as many sessions as possible so long as the server is not overloaded. Our previous work on admission control based on predictable service time [8] could serve this purpose. In addition, the scheduling algorithm takes care of the situation when the admitted sessions are beyond the server's capacity. It seeks the best scheduling that produces as many completed sessions (not necessarily requests) as possible.

In the context of sessions, each of the waiting queue represents a particular task of the session sequence and its output serves as input to the other queues. So proper shaping of these queues by means of priority scheduling among different queues can alleviate overload conditions. This is the basic idea of our scheduling algorithm.

Another phenomenon that is frequently observed in web servers is that during overload, more number of tasks gets aborted. Before abortion, these tasks consume excessive system resources during the crunch period. To resolve this ill-effect we use a scheme in which requests of sessions that have a higher probability of getting completed are scheduled first. Such a scheduling approach helps the server in doing more useful work during overload situations, while avoiding the service of requests whose sessions are likely to get aborted.

4.1. Comparison of scheduling algorithms

The popular scheduling algorithms that are used in web servers include round robin (RR), earliest deadline first (EDF) and weighted fair sharing (WFS). RR and EDF do not consider the relationship of inter-session request transition thus they cannot help in session-based overload control. WFS provides higher levels of service to the tasks that have higher priority. Thus the queue length accumulation at some stages can reach a steady-state by lowering the request injection rate and raising the service rate. However, this discrimination increases request accumulations at other queues, which in turn would result in request drops because of timeout. This domino effect disrupts the normal request transition flow into other queues and eventually leads to lower throughput in terms of the number of completed sessions.

We introduce a measure called *server's productivity*, which is defined as a function of request completion and error rate of requests that belong to ongoing sessions for a server, and can be formally expressed as follows.

> Server's productivity during time interval T: If the number of requests completed within T is c, and the number of requests aborted during this time is e, then the *server's productivity* during T is $(c - e)$.

Server productivity is thus a measure of the amount of useful work done by a server during a given time period. Serving fewer requests while more number of requests get aborted leads to a negative productivity.

A request abortion occurs either because of some of the internal problems at the server side or due to the processing timeout imposed by the script languages like ASP and PHP. A request could also be aborted by the user because of impatience. Ignoring the internal problems of the server, we use request timeout to represent all the server processing failures. To illustrate how the weight assignment could affect server productivity, we simulated a round robin scheduler and collected the results under different weight assignments. The simulator has four queues corresponding the four states (MM, CO, SR, BR) with the same transition probability as was listed in Fig. 5. The processing times of the stages are 0.5, 1, 1.5, and 2, respectively. The timeout duration was assumed 20 time units and the simulation duration was set to 10,000 time units. The more weight a queue is assigned, the more CPU time it can use. Table 2 displays the different server productivities. It is observed that proper weight assignment can significantly improve the overall

Table 2
Server's productivity comparison

Weight assignment	Requests completed	Requests timed out	Server's productivity
(1,1,1,1)	3077	747	2330
(5,1,1,1)	3930	616	3314
(1,5,1,1)	1585	826	759
(1,1,5,1)	2520	880	1640
(1,1,1,5)	1685	879	806
(5,1,5,1)	3463	477	2986
(1,5,1,5)	1161	767	394

performance by increasing the number of completed requests and reducing the number of timeouts. In this set of results, the best case performance is nearly eight times better than the worst case. This inference inspires to seek productivity improvement through appropriate weight assignment.

Motivated by the server productivity study, we propose a DWFS scheme based on the temporal relationship in web session in such a manner that the weight distribution is not static all the time. Instead, it depends on the accumulation at the queue and the output rate with the goal to improve the server productivity. Unlike the traditional algorithms that seek short term throughput improvement, DWFS tries to smooth the domino effect of overloads in pursuit of sustained throughput.

4.2. Dynamic weighted fair sharing

The objective of DWFS is to improve the server's productivity through dynamic weight assignment for scheduling purpose. In an overloaded server, processing at one queue is not productive when it overwhelms other queues. For example, in an e-commerce server, the merchandise browsing and checkout are two queues. If the browsing queue is processed faster than the checkout queue, then the clients proceed to check out only to be jammed, and most of the requests get timed-out. In this case the browsing queue becomes unproductive. On the other hand, if the server distributes more weight on the check out queue such that the requests in the browsing queue will experience longer but tolerable service time, then the input to the downstream checkout queue is reduced and their probability of receiving service is increased, consequently the end user can perceive a faster service for the entire session.

The other aspect of DWFS is that, if a server knows a priori that the request it is serving is unproductive, it can stop or delay processing the current request queue and transfer the weight to serve other queues to improve the server's productivity.

The modeling tool used for DWFS is a queuing network with limited waiting room. A k-waiting room queue can accommodate at most k entries waiting for service and the new arrivals are simply dropped. More specifically, as a measure to allow dynamic weighing, k is defined to be the ratio of service time and session timeout period. Each output from a queue produces a credit if the output goes to a queue that is not yet full (productive queues). The credit reflects the productiveness of the service. If service to a queue is known to be unproductive, then its weight is transferred to handle other queues. This conflicts with philosophies behind some other scheduling algorithms that seek maximal throughput from the server in all situations. In DWFS, some jobs may be dropped or delayed even though they could have been served before the deadline if more weight were assigned to them. However, in the long run, more incoming requests can meet their service expectation thus the overall throughput increases. If the drop rate is not high to overshadow the throughput, the server's productivity improves.

Productivity function:

$$f(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} P_{i,j} * 1\left(L_j * \frac{T_j}{w_j} < \text{Timeout}\right), \tag{5}$$

where w_j is the normalized weight assigned to queue j and L_j is the queue length. $1(x) = 1$ if x is true, 0 otherwise.

The productivity function defined in Eq. (5) states that if the output from a queue is served before the deadline, then it is considered productive. The productiveness of the output is determined by the destination it leaves for; if the destination queue is full, then no credit is earned but a penalty is imposed, otherwise a credit is added. The credit can also be inversely proportional to the queue length to avoid filling up the queue. Since there are more than one possible destinations for the output, the credit is reflected by multiplying it with the transition probability. The capacity of the waiting room depends on the service weight; the more weight assigned to the queue, the less time a request takes to complete, and more requests can be served before the timeout period expires.

Eq. (5) can be rewritten as

$$f(n) = \sum_{j=1}^{n} Q_j * 1\left(L_j * \frac{T_j}{w_j} < \text{Timeout}\right), \tag{6}$$

where $Q_j = \sum_{i=1}^{n} P_{i,j}$. The maximization of $f(n)$ can be achieved through dynamic programming method as illustrated in Algorithm 1, the complexity of which is $O(n^2)$. When the number of stages is small, the computational overhead is trivial.

Algorithm 1. Pseduocode of productivity function solver

- INPUT:
```
L[1..n], T[1..n], Q[1..n], and timeout have
the same meaning as in Eq. (5)
w: remaining weight
j: current queue to calculate
```
- OUTPUT:
```
maximum productivity
```
- ALGORITHM:
```
int max_fn(w, j) {
  if(j >= n)
    return 0;
  min_w = T[j] * L[j]/timeout;
  if(min_w > w)
    return max_fn(w, j + 1);
  f1 = max_fn(w-min_w, j + 1) + Q[j];
  f2 = max_fn(w, j + 1);
  return max(f1, f2);
}
```

It is inferred from the productivity function that, when other parameters are fixed, the timeout value determines how many jobs can be queued in each stage. Bigger the timeout value, the more number of jobs that can be queued leading to longer mean queueing time. We were thus inspired to differentiate QoS based on timeout values. This impact is further explored in Section 5.

5. Experimental performance evaluation

From the above discussion, we can see that the DWFS's approach to relieve an overloaded server is to add weights on those requests whose descendant requests within the same session can be honored. Those requests whose descendant requests are predicted to miss their delay bound are delayed for later processing. To verify the feasibility of this scheme, we implemented the algorithm in an Apache web server and evaluated its performance.

5.1. Experimental setup

The test-bed contains a web server and several clients. The server has an Intel PIII 733MHZ CPU and 128 MB RAM, running Apache 1.3.19 for MS Windows 2000. Apache [3] is an open source, widely used web server. Fig. 7 illustrates the request handling process in the Apache server. At runtime, the Apache server consists of worker processes (or threads in some systems like MS Windows) and one listener. The listener listens on HTTP port (usually TCP 80) and accepts new connections. It adds the connections into a job queue by calling add_job( ). At this time, add_job( ) does not parse the HTTP request lines. Each of the worker processes unlinks a job from the job queue by calling remove_job( ). Only then the request line is parsed and the priority group and stage the job belongs to are known. This working mechanism does not fit DWFS in the sense that the worker processes have no control over the job queues. We have changed the way new jobs are added.

As shown in Fig. 8, instead of connections, in our modification, the jobs are HTTP requests and

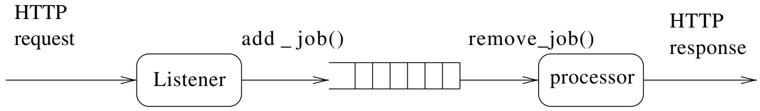

Fig. 7. Apache implementation of request handling

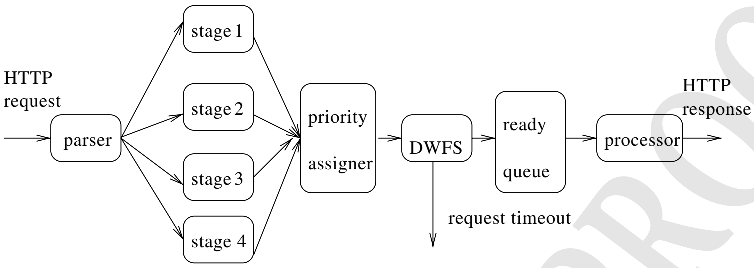

Fig. 8. Revised implementation of request handling

the parser sorts them into different queues based on their URL's and records the time-stamp they are added into the queue. Then the priority assigner assigns different priority to the requests based on their IP addresses. The DWFS module is invoked when the ready queue is empty to exercise the DWFS algorithm to assign jobs to the worker processes and drops the requests that are timed out. Since Apache server itself has no control over the CPU time slot, the weights are assigned as the number of requests from the queues to be dispatched to the ready queue. At this time, the request that exceed the timeout are dropped. Finally the processor handles the requests within the ready queue.

The clients used in the test-bed are several Sun UltraSparc 10 workstations running Solaris 2.8. The benchmarking tools used in the experiment is a modified version of WebStone 2.5 [22]. The request distribution in the original WebStone benchmark is not adequate for our purpose. The way a client picks a URL is randomized and the URL array size is too small in the original benchmark. We modified the source so that each client sequentially picks a URL from the array to simulate a session and all the clients together replay the request trace of the online retailer that we have characterized earlier.

The working mechanism of the experiment is as follows. A master process instructs a configurable number of clients to send HTTP requests to a web server for a specified time period. During the test, the clients choose a URL from an array and keep track of the connection time, response time, and other parameters. After the test finishes, the clients send the statistics to the webmaster process and the latter reports the test results after collecting all the clients' data. The final results contain information such as server connection rate (the number of connections the server accepts; the higher the better), average response time (the average timespan the client receives the whole response; the smaller the better), average throughput (the server' throughput during the test time; the higher the better). There are other HTTP benchmarking tools available like httperf [16], SpecWeb99 [20], SURGE [6], etc. But WebStone's ability to control the number of clients and URL array and its excellent result reporting tools was appropriate and the modified version was adequate for our experiment.

We created over 4,000 files in the web server and divided them into four queues as stated in the previous section. The average processing time of URL's in each queue under normal server load is 0.5, 20, 10 and 5 s respectively. In the test, the WebStone clients were instructed to replay the trace of the online retailer's server collected during the period 20:00–21:00.

COMPNW 2747 ARTICLE IN PRESS No. of Pages 15, DTD = 4.3.1
4 February 2003 Disk used SPS-N, Chennai

5.2. Experimental results

From the test, we found that some of the performance parameters such as server connection rate, number of completed requests and server throughput (in terms of the number of requests served) were more or less the same, while the average response time varied significantly. For better comparison, we have chosen the response time as the primary performance measure for our analysis. We also analyzed parameters such as request timeout rate due to DWFS, the queueing time, and the processing time of each request from the server log at the web server side. We observed that the web server was overloaded when the number of clients reached 40 and essentially became saturated after being requested by 60 clients, so the following results and discussions are focused on the results with the number of clients as 40, 50, and 60.

The experiments were conducted under different DWFS session timeout settings and the results were compared to the original Apache server with the same configurations. In the DWFS tests, the clients were evenly divided into three priority classes. Each priority class has a scheduling timeout, the shorter timeout a request is assigned, the higher is its priority and thus it gets quicker service. In WebStone, the webmaster process always tries to evenly divide the number of total processes on the client workstations, making the number of processes in each of the priority class approximately equal.

Tables 3–5 show the DWFS results with different timeout values, and Table 6 shows the results from the unmodified Apache server with the same configuration. Fig. 9 depicts the response time comparison of the four configurations. In Tables 3–5, the values in *Number of Clients* is represented as an fraction because the number of processes in each priority class varies in each test iteration but is approximately one-third of the total number of clients. Since the original Apache server does not drop requests because of timeout, only the average response time is listed in Table 6.

It is observed from the tables that, DWFS can significantly improve server performance by reducing the response time up to 52%. The smaller

Table 3
DWFS with session timeout period of 10 s

Number of clients	Average response time (s)	Request timeout rate (%)
40/3	6.902	0.16
50/3	7.152	0.28
60/3	6.968	0.44

Table 4
DWFS with session timeout period of 15 s

Number of clients	Average response time (s)	Request timeout rate (%)
40/3	8.537	0.11
50/3	8.357	0.20
60/3	8.772	0.35

Table 5
DWFS with session timeout period of 20 s

Number of lients	Average response time (s)	Request timeout rate (%)
40/3	9.512	0.01
50/3	9.864	0.16
60/3	10.584	0.24

Table 6
Response time behavior of the Apache server

Number of clients	Average response time (s)
40	9.623
50	11.885
60	14.267

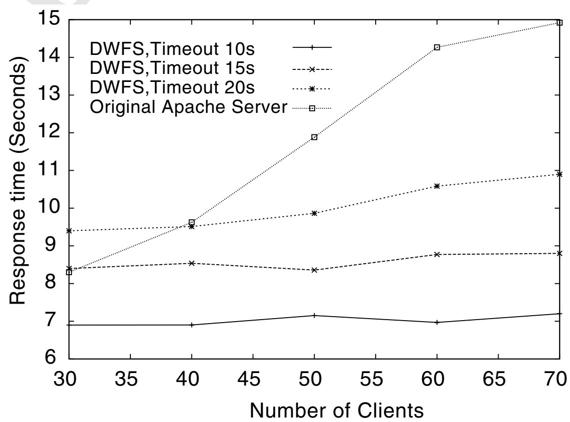

Fig. 9. Response time comparison.

the session timeout, the shorter is the response time and thus better is the service. DWFS incurs additional request timeout. The timeout rate rises significantly with respect to the number of clients and the timeout value. But such occurrence is less than 0.5%, the impact of which is insignificant on the service availability provided by an overloaded server. This observation also distinguishes DWFS from the shortest-job-first scheduling in the sense that the shortest-job-first scheduling starves long jobs which may lead to more request timeouts. Finally, the relatively steep slope of the original Apache server curve compared to DWFS curves in Fig. 9 reveals that the server performance using DWFS is more scalable when the number of the clients increases.

To investigate the underlying factors of the response time difference, we further analyzed the anatomy of the response time. We divided the response time into queueing time and processing time. Queueing time is the period during which a request remains queued and processing time is the interval between when the request was read and the time when the HTTP response sent. In Apache architecture, the queueing time starts when a job is accepted and ends at the point when the request starts getting processed by a worker thread; its processing time starts at that point. For a DWFS enabled Apache server, however, since every request line is read immediately after it has been accepted, the queueing time spans from the time the request is read to the moment when a worker thread begins to process it. Fig. 10 presents the anatomy of the response time under different configurations. The x-axis label specifies server type/timeout value/number of clients. The timeout value is zero for the original Apache server because it has no scheduling restraints.

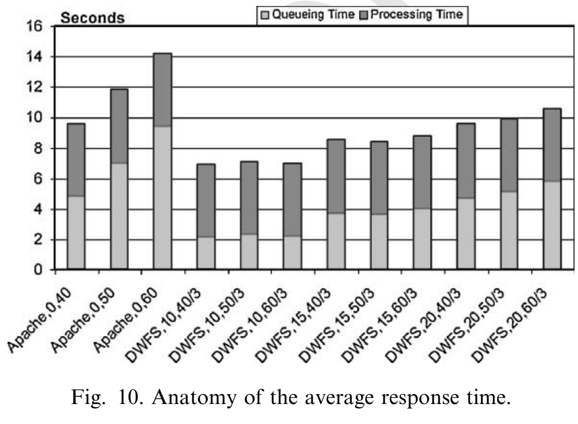

Fig. 10. Anatomy of the average response time.

From Fig. 10, it is observed that the queueing time of the original Apache server is dependent on the number of the clients. The more the number of clients, the longer is the queueing time. We believe that this effect is a direct result of its best effort scheduling discipline, where requests are queued on a first-come-first-serve manner and short requests have to wait for the completion of long requests even if these long requests' turnaround time may exceed the delay bound and be aborted by the impatient clients. For the DWFS's case, the queueing time also varies for different timeout settings but is much shorter than those of the original Apache server. Under the same timeout setting (thus the same priority class), the queueing time remains stable, which in turn means that the session level serviceability is maintained. This is due to the pre-emptive scheduling and the early dropping of those requests whose pre-assigned service time cannot be guaranteed for the descendant requests within the same sessions.

In Section 4.2, we claimed that by varying the timeout value, the number of requests in each queue will change. For bigger timeout value, more requests reside in the queues, as a result of which the mean queueing time increases. This was found to be true in the experiment. Figs. 11 and 12 plot the queue length under timeout value 10 and 20 s with 40 clients. Table 7 lists the average request latency at each queue using different timeout values. It is observed that queue lengths under timeout 20 s are bigger that those in 10 s and queue latencies under timeout 20 s are correspondingly longer. These results verify that varying timeout value can provide service differentiation.

Finally, we compared the session abortion rate under different configurations. We assume that a session is aborted if one of the requests within it does not receive service before its delay bound. In the real world, when this occurs, the end users get impatient and abort the session. In DWFS, a session gets aborted when the request has not been processed before its timeout value. While there is no timeout constraint in Apache, we assume that

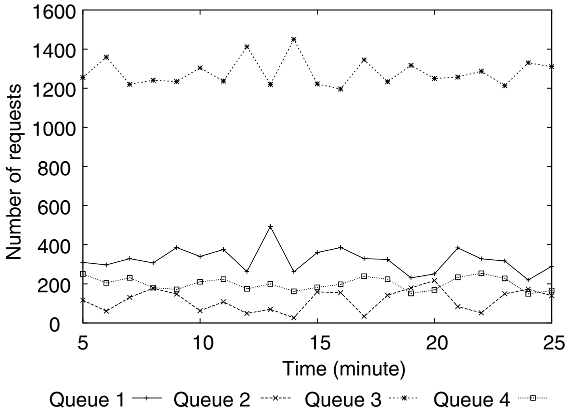

Fig. 11. Queue length at timeout 10 s.

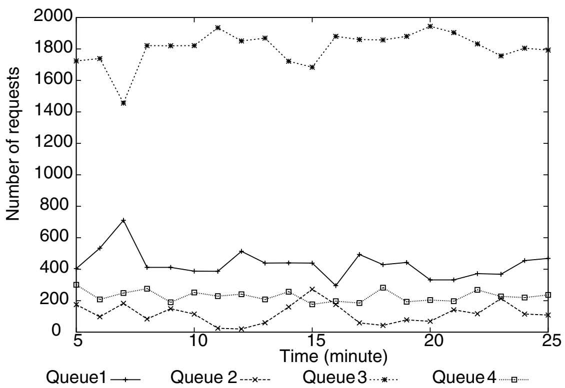

Fig. 12. Queue length at timeout 20 s.

Table 7
Average latency

Queue	Timeout 10 s	Timeout 20 s
1	1.6	2.3
2	27.5	27.9
3	14.5	20.9
4	6.0	6.6

every session has a timeout value of 15 s. Fig. 13 depicts the comparison. It is observed that for DWFS, the aborted session rate is relatively small and the priority class with longer timeout value usually has lower abortion rate. For Apache server, the aborted session rate is sensitive to the number of clients which means more and more sessions get aborted as the server load increases.

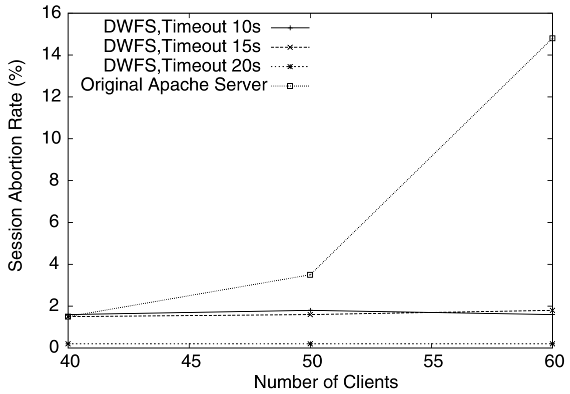

Fig. 13. Session abortion rate comparison.

6. Related work

A limited number of work has been reported on sessions characterization in web servers. In [4], the authors provide parameters based on the World Cup 98 server log, which include session length, inter-session time, and their implication on server performance. Our characterization work provide complementary results on workload composition, session stage transition, and the ratio of request arrival rate to session generation rate. These empirical results can be exploited to recognize user browsing behavior and capacity planning.

In the context of capacity planning, [13] provides a model based on bandwidth demands for memory, processors data bus, NIC and I/O buses. It is practical for server configuration. Our capacity planning model is targeted towards session level SLA specification and overload prevention.

Although a plethora of work on web servers have addressed performance issues in web servers, the studies on overload control has been limited. An approach for overload control by content adaptation has been proposed in [1]. Under high load the servers resorts to low fidelity images that consume less system resources, thus reducing the load. Content adaptation is applicable mainly to static web content. Overload control using operating system support has been studied in [5,15,21]. The server behavior under overload has been analyzed in [15] and three solutions are proposed to help relieve an overloaded server. These solutions include direct control over kernel timeouts and resource limits, resource introspection, and disaster management. Three kernel-based mechanism that prevent server from being overloaded by admission control and service differentiation are presented in [21]. Their mechanisms include TCP SYN policing that control the TCP connection rate, prioritized listen queue and HTTP header-based connection control that provides service differentiation. A new kernel facility called resource container which can effectively audit overall resource usage by each process is presented in [5]. This scheme is useful for service differentiation as well as overload control. In [19], the authors have studied web server overload control through three different schemes. The first approach is based on the network interface level request dropping. The second approach refers to a feedback mechanism from the application level to throttle the traffic volume. The third approach is a hybrid of the other two schemes. These schemes significantly improve server throughput under high load. All these solutions have not considered the session integrity and hence have limited applications for session-based web traffic.

Most of the prior work on overload control having examined performance on per request basis, which may not be adequate for many applications that require session-based overload control. A session-based admission control scheme has been reported in [9], which prevents overload by efficient admission control. They monitor the server load periodically and estimate the load in near future. If the predicted load is higher than a predefined threshold, no new requests are admitted. This situation may lead to denial of services. The proposed DWFS scheme is targeted for efficient scheduling of requests and complements the work reported in [9] in maintaining long term server availability.

7. Conclusion

Overload control ensures service availability in varying workload and is an indispensable part of network server engineering. This paper presents QoS capacity planning and scheduling algorithm for overload control based on characterization of a commercial web server log. The main idea of the proposed scheme is to use session-based overload control. Performance measures of web services in terms of sessions is more meaningful than the measures based on individual requests. We have targeted QoS-aware web servers that provide guaranteed QoS based on the requirement of sessions. The traffic conformation function provides quantitative solution for SLA specification and can be used in commercial servers. We have proposed and evaluated a new scheduling algorithm called DWFS, which discriminates the scheduling of requests on the basis of the probability of completion of the session that the requests belong to. The proposed scheduling algorithm improves server productivity under heavy load by more than 50% in the configuration studied in this paper. This work can be used as a framework for further development and deployment of session-based overload control techniques.

References

[1] T. Abdelzaher, N. Bhatti, Web content adaptation to improve server overload behavior, in: Proceedings of the 8th WWW Conference, Toronto, Canada, May 1999.

[2] Akamai Corporation, http://www.akamai.com.

[3] Apache Group, http://www.apache.org.

[4] M. Arlitt, Characterizing web user sessions, in: Proceedings of the Performance and Architecture of Web Servers Workshop, Santa Clara, CA, June 2000.

[5] G. Banga, P. Druschel, J. Mogul, Resource containers: a new facility for resource management in server systems, in: Proceedings of OSID, February 1999.

[6] P. Barford, M. Crovella, Generating representative web workloads for network and server performance evaluation, in: Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Madison, WI, June, 1998.

[7] J. Borges, M. Levene, Data mining of user navigation patterns, in: Proceedings of the Workshop on Web Usage Analysis and User Profiling, August 1999, San Diego, CA, USA.

[8] X. Chen, P. Mohapatra, H. Chen, An admission control scheme for predictable server response time for web accesses, in: Proceedings of the 10th WWW Conference, Hong Kong, May 2001.

[9] L. Cherkasova, P. Phaal, Session-based admission control-a mechanism for improving performance of commercial web servers, in: Proceedings of IEEE/IFIP IWQoS 99.

[10] M.E. Crovell, A. Bestavros, Self-similarity in world wide web traffic: evidence and possible causes, IEEE/ACM Transactions on Networking (December) (1997).

[11] Internet Engineering Task Force, http://www.ietf.org.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, RFC 2616: Hypertext Transfer Protocol–HTTP/1.1, http://www.ics.uci.edu/pub/ietf/http/rfc-2616.txt.

[13] K. Kant, Y. Won, Server capacity planning for web traffic workload, IEEE Transactions on Knowledge and Data Engineering 11 (5) (1999).

[14] D.M. Kristol, L. Montulli, Internet draft: HTTP state management mechanism, http://www.ics.uci.edu/pub/ietf/http/draft-ietf-http-state-man-mec-12.txt.

[15] J.C. Mogul, Operating system support for busy internet services, in: Proceedings of the 5th Workshop on Hot Topics in Operating Systems, Orcas Island, WA USA, May, 1995.

[16] D. Mosberger, T. Jin, httperf—A tool for measuring web server performance, in: Workshop on Internet Server Performance, Madison, WI, USA, June, 1998.

[17] J. Pitkow, P. Pirolli, Mining longest repeating subsequences to predict world wide web surfing, in: USENIX Symposium on Internet Technologies and Systems, Boulder, CO, 1999.

[18] Logging properties reference, http://www.windows.com/windows2000/en/server/iis/html/core/iiintlg.html.

[19] V. Tewari, R. Iyer, K. Kant, Overload control mechanisms for web servers, in: Performance and QoS of Next Generation Network, Nagoya, Japan, November, 2000.

[20] The Standard Performance Evaluation Corporation, http://www.spec.org.

[21] T. Voigt, R. Tewari, D. Freimuth, A. Mehra, Kernel mechanisms for service differentiation in overloaded web servers, in: Proceedings of the 2001 USENIX Annual Technical Conference, Boston, MA, USA, June, 2001.

[22] WebStone, http://www.mindcraft.com/webstone.

[23] H. Zhu, H. Tang, T. Yang, Demand-driven service differentiation in cluster-based network Servers, in: Proceedings of the IEEE INFOCOM 2001, Anchorage, April, 2001.

Huamin Chen received the B.S. and M.S. degrees from Huazhong University of Science and Technology, Wuhan, China in 1996 and 1999, respectively. He is working towards the Ph.D. degree in the department of Computer Science, University of California at Davis. His research interests are in Internet server performance improvement and QoS provisioning, Web Services, computer networks, and operating system optimization.

Professor Prasant Mohapatra is currently an Associate Professor in the Department of Computer Science at the University of California, Davis. In the past, he was on the faculty at Iowa State University and Michigan State University. He has also held Visiting Scientist positions at Intel Corporation and Panasonic Technologies. Dr. Mohapatra received his Ph.D. in Computer Engineering from the Pennsylvania State University in 1993. Dr. Mohapatra is currently on the editorial board of the IEEE Transactions on Computers and has been on the program/organizational committees of several international conferences. He was the Program Chair for the PAWS Workshop during 2000 and 2001, and the Program Vice-Chair for the International Conference on Parallel Processing, 2001. He is also the Co-Editor of the January 2003 issue of IEEE Network.

Dr. Mohapatra's research interests are in the areas of wireless mobile networks, Internet protocols and QoS, and Internet servers. Dr. Mohapatra's research has been funded though grants from the National Science Foundation, Intel Corporation, Panasonic Technologies, Hewlett Packard, and EMC Corporation.