ADAPTLOAD: effective balancing in clustered web servers under transient load conditions*

Alma Riska   Wei Sun   Evgenia Smirni   Gianfranco Ciardo
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187-8795, USA
{riska,wsun,esmirni,ciardo}@cs.wm.edu

Abstract

We focus on adaptive policies for load balancing in clustered web servers, based on the size distribution of the requested documents. The proposed scheduling policy, ADAPTLOAD*, adapts its balancing parameters on-the-fly, according to changes in the behavior of the customer population such as fluctuations in the intensity of arrivals or document popularity. Detailed performance comparisons via simulation using traces from the 1998 World Cup show that* ADAPTLOAD *is robust as it consistently outperforms traditional load balancing policies, especially under conditions of transient overload.*

Keywords: *clustered web servers; load balance; adaptive policies, transient overload.*

1 Introduction

Clustering with a single system image, where a set of hosts behaves as a single host, is a popular solution to meet the need for web site scalability and availability with respect to an ever-increasing customer population. Additionally, rapid fluctuations in customer demands triggered by unpredictable events, such as breaking news in a news web server or special sales in an e-commerce site, makes the ability to swiftly react to sudden system overload an important but remarkably difficult task. Consequently, inherent to the commercial success of web sites is the ability to continuously provide high performance and availability in a cost-effective manner [2].

In this paper we concentrate on effective load sharing for a clustered web server, whose abstraction is illustrated in Figure 1. We consider a web site that uses multiple servers housed in the same geographic location and achieves end-user transparency by making multiple hosts behave like a single host. Such clustered designs can be implemented at various levels: DNS-based, dispatcher-based (at the network level), and server-based [5, 6]. Scheduling the incoming load in such systems is fundamental for high performance.

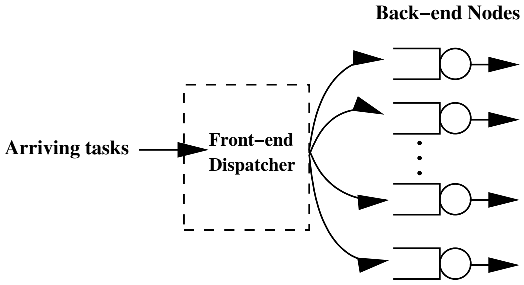

Figure 1. Model of a clustered web server.

A robust scheduling policy must carefully take into account both the arrival process of incoming requests and the distribution of their service requirements. Any observed burstiness in the average number of arrivals per unit time must trigger a change in the policy parameters, so that it adapts to the new arrival pattern. Additionally, there is growing evidence that the distribution of web document sizes does not fall into the "well-behaved" class of exponential distributions but is instead heavy-tailed [1, 3, 4]. The complexity of the arrival and service requirements to the system makes effective request scheduling a difficult task.

A significant body of research in load balancing has been developed over the years, but it shares the basic assumption that the service requirements of jobs follow an exponential distribution. Thus, these traditional load balancing policies fail to balance the load if the workload is heavy-tailed [8]. Instead, for heavy-tailed workloads, there is an increasing trend in devising load balancing policies that strive to avoid assigning "long" and "short" jobs to the same server [7, 8]. This trend is fueled by ample empirical evidence suggest-

*This work was partially supported by the National Science Foundation under grants EIA-9974992 and CCR-0098278.

ing that the sizes of web documents (and consequently their service demands) follow a heavy-tail distribution [1, 3, 4]. In web server systems, fortunately, the size of the file corresponding to a given URL request is a good characterization of the length of the job, so approaching this ideal is not out of the question: this is the reason for the popularity of size-based policies for clustered web server systems [8].

In [7], we proposed EQUILOAD, a new scheduling policy that balances the load in the system by using the size of each incoming request to determine the identity of the back-end server that will satisfy it, without considering any information regarding the load of the individual back-end servers. In other words, EQUILOAD advocates dedicating servers to requests of similar sizes. Assuming that there are N back-end servers, EQUILOAD requires partitioning the request sizes into N intervals, $[s_0 \equiv 0, s_1)$, $[s_1, s_2)$, $\ldots$, $[s_{N-1}, s_N \equiv \infty)$, so that server i will be responsible for satisfying requests of size between s_{i-1} and s_i. Although the policy is based solely on the distribution of the incoming task sizes, it manages to load each server with approximately the same amount of work, so that they are equally utilized; even more important from a user's perspective, EQUILOAD also results in lower expected job slowdown[1], since short jobs are effectively separated from long jobs; finally, as a natural side-effect, EQUILOAD maximizes cache hits at the back-end servers [7] by behaving similarly to a "locality-aware" allocation policy [9].

Exact *a priori* knowledge of the request size distribution is an essential requirement for EQUILOAD performance, and its main weakness as well, since such knowledge may not always be available. For example, special events may drastically alter the relative popularity of the web server documents or result in new documents being offered. In this paper, we leverage EQUILOAD by providing an on-line mechanism to *continuously adapt* the policy parameters to changes in the incoming request pattern. The new policy we propose, called ADAPTLOAD, dynamically re-adjusts its parameters based on system monitoring and on-the-fly characterization of the incoming workload.

We evaluate ADAPTLOAD via trace-driven simulation using real trace data from the 1998 World Cup Soccer web site. The selected trace data contains significant fluctuations in the workload arrival pattern, thus is a good means to evaluate the performance of a policy that strives to adapt to changing workload behavior. Our simulations indicate that knowledge of a finite portion of the past workload can be used as a good indicator of future behavior. Using a geometrically discounted history of the workload starting from the immediate past, we tune the values of the $N-1$ interval boundaries and show that an intelligent adaptation of these boundaries is not only feasible but it can also provide excellent performance.

This paper is organized as follows. Section 2 contains a characterization of the workload used to drive our simulations. Section 3 presents our proposed load balancing policy, ADAPTLOAD. Finally, Section 4 gives a detailed performance analysis of ADAPTLOAD and proposes ways to better parameterize it. Section 5 concludes the paper.

2 The workload

An essential characteristic of an effective load balancing policy is the ability to minimize the expected job slowdown. As the policies proposed in this paper adjust their parameters exclusively according to the distribution of the incoming workload, detailed knowledge of the workload statistical characteristics is essential to policy effectiveness. Thus, we first present the salient characteristics of the workload used in our analysis, the traces of the 1998 World Soccer Cup web site obtained from the Internet Traffic Archive[2]. The server for this site was composed of 30 low-latency platforms distributed across four physical locations.

The traces provide information about each request received by the site during 92 days, from April 26, 1998 to July 26, 1998. Trace data were collected for each day during the total period of time that the web server was operational. Since the focus of this work is on load balancing irrespective of possible caching policies at the server, we only extract arrival and service information for each trace entry, i.e., the date and time of each request as well as the content length of the transferred document from each trace record. Furthermore, we assume that the time to serve a request is linear in the size of the transferred document.

Analysis of the unique file size distribution across all 92 days indicated that the workload is heavy-tailed. This trend persists if the workload is also analyzed on a day by day basis [7, 10]. For a detailed analysis of the World Cup workload see [1]. The existence of heavy-tailed service time distribution implies that caution should be exercised when routing requests to the back-end servers: small files should be separated from large ones to avoid having the former "getting stuck" in the queue behind the latter. It is therefore imperative to know not just the distribution of the workload requests, but also if and how this distribution changes during the course of the day.

Figure 2 focuses on the arrival and service characteristics of eight "busy" days with high total traffic intensity for the 1998 World Cup, namely from June 23 1998 to June 30, 1998. In all four plots, data is collected at 5-minute intervals. Figure 2(a) illustrates the arrival rate into the system as a function of time and shows that there is a clear periodic pattern on a per-day basis. Six out of the eight days show a sharp increase (two spikes) in the arrival intensity during

[1] Slowdown, the ratio of response time to service time for a job, is a fundamental measure of responsiveness in web servers, since users are more willing to wait for "large" requests than for "small" ones.

[2] Available at `http://ita.ee.lbl.gov/`.

the evening hours, a direct outcome of posting the results of distinct soccer games. This effect is observed also during the other two days, June 27 and June 28, although not as markedly as with the rest of the selected days. Figure 2(b) illustrates the total volume of transferred data for the same 8-day period.

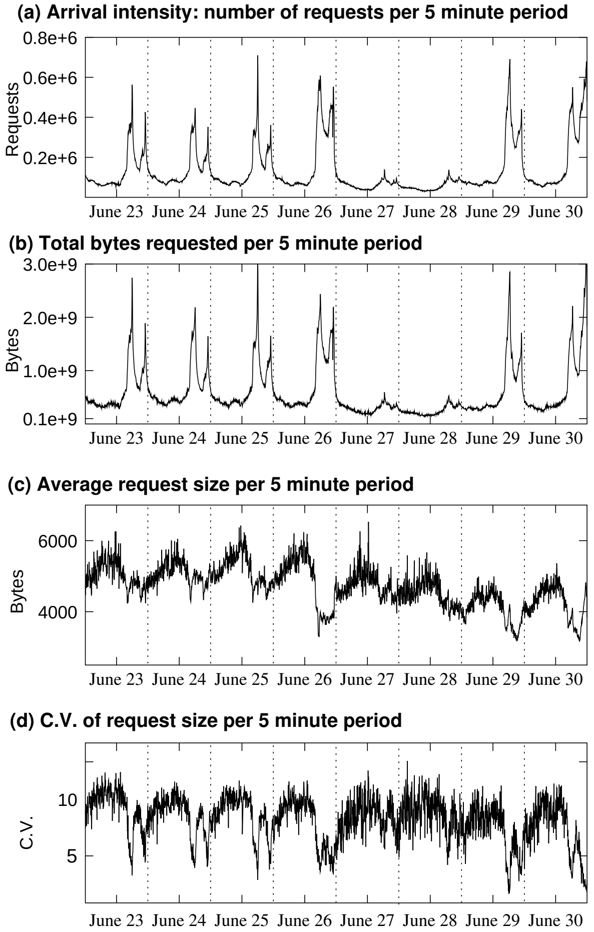

Figure 2. Arrival and service characteristics of a "busy" week of the 1998 World Cup trace.

To investigate the correlation between the arrival intensity and the size of each request, we plot the average request size experienced by the site during the same period, and observe that there is negative correlation between the arrival intensity and the request size: peaks in Figures 2(a) and (b) correspond to dips in Figure 2(c), suggesting that the most popular files during the busy evening periods have a small size. Figure 2(d) plots the coefficient of variation of the request size, and further confirms that the requests during the high arrival intensity periods are more uniform in size than the rest of the day. However, coefficients of variation as high as ten indicate the presence of heavy-tails.

This analysis illustrates the difficulty of policy parameterization. The policy parameters need to swiftly adapt to changes in the request distribution, which can vary dramatically from an hour to the next within the course of a day. Consequently, these parameters must be tuned carefully, especially when high arrival intensities are expected.

3 ADAPTLOAD: on-line load balancing

EQUILOAD, first proposed in [7], is based on the observation that directing tasks of similar size to the same server reduces the slowdown in a web server [10]. In a cluster of N web servers, EQUILOAD requires partitioning the possible request sizes into N intervals, $[s_0 \equiv 0, s_1)$, $[s_1, s_2)$, up to $[s_{N-1}, s_N \equiv \infty)$, so that server i, $1 \leq i \leq N$, is responsible for satisfying the requests with size falling in the i^{th} interval.

The value of the $N-1$ size boundaries $s_1, s_2, \ldots, s_{N-1}$ is critical, as it determines the load seen by each server. The choice of boundaries should result in a uniform expected slowdown at each server, by providing each back-end server with (approximately) the same load. Thus each interval $[s_{i-1}, s_i)$ should be set so that the requests routed to server i contribute a fraction $1/N$ to the value of the expected request size, that is, for $1 \leq i \leq N$,

$$\int_{s_{i-1}}^{s_i} x \cdot dF(x) \approx \frac{1}{N} \int_0^{\infty} x \cdot dF(x) = \frac{\overline{S}}{N},$$

where $F(x)$ is the CDF of the request sizes. Given a trace of R requests and their sizes, the s_i boundaries can be determined using the algorithm outlined in Figure 3. EQUILOAD builds a discrete data histogram (DDH) encoding the empirical size distribution of the trace requests[3]. This remarkably simple mechanism to determine the s_i boundaries maximizes the homogeneity of request sizes assigned to each back-end server, thus it improves the overall responsiveness of the system. Furthermore, it does not require any knowledge of the load at the servers.

3.1 Learning from the immediate history

EQUILOAD provides a robust scheduling solution for heavy-tailed workloads and consistently outperforms classic load-balancing policies such as the Join Shortest Queue policy [7]. However, EQUILOAD requires in principle *a priori* knowledge of the request size distribution experienced by the web server. In Section 2, we discussed how the workload can be highly *variable* across not only successive days but also within a single day. Therefore, a dynamic on-the-fly adjustment of the s_i boundaries is imperative for high performance.

One simple solution to the above problem is to use the system history, more precisely the last K requests seen

[3]We can think of the DDH as a vector of (b, c) pairs, where b is a particular size of requests encountered in the trace (in bytes), c counts the number of times requests of this size appear in the trace, and the vector entries are sorted by increasing values of b.

1. compute the DDH of the sizes of the R requests for F different files: $\{(b_f, c_f) : 1 \le f \le F\}$
2. compute the total requested bytes: $B \leftarrow \sum_{f=1}^{F} c_f b_f$
3. initialize the accumulated size and DDH index: $A \leftarrow 0$ and $f \leftarrow 0$
4. for $i = 1$ to $N-1$ compute $s_1, s_2, \ldots, s_{N-1}$ by scanning the DDH:
  a. while $A < B \cdot i/N$ do
    I. increment A by the contribution of entry f of the DDH: $A \leftarrow A + c_f b_f$
    II. move to the next entry of the DDH: $f \leftarrow f + 1$
  b. set the i^{th} boundary so that $\sum_{f=1}^{s_i} c_f b_f = B \cdot i/N$: $s_i \leftarrow f$

Figure 3. Setting the boundaries $s_1, s_2, \ldots, s_{N-1}$ with EQUILOAD.

by the system, to build the DDH needed to determine the boundaries for the allocation of the next K requests. K must be chosen wisely: it should be neither too small (since it must ensure that the computed DDH is statistically significant) nor too large (since it must allow adapting to workload fluctuations). In Section 4 we provide a more refined algorithm to use the history in a geometrically-discounted fashion.

3.2 Introducing fuzzy boundaries

An additional algorithmic modification is necessary to ensure good performance, given that the boundaries are computed using an empirical distribution. If a significant portion of the workload consists of a few popular files, it may not be possible to select N *distinct* boundaries and still ensure that each interval $[s_{i-1}, s_i)$ corresponds to a fraction $\frac{1}{N}$ of the load.

To solve this problem, we introduce "fuzzy" boundaries. Formally, we associate a probability p_i to every "fuzzy" boundary point s_i, for $i = 1, \cdots, N-1$, expressing the portion of the requests for files of size s_i that is to be served by server i. The remaining portion $1 - p_i$ of requests for this file size is served by server $i+1$, or even additional servers (for the 1998 World Cup workload, sometimes we had to extend a fuzzy boundary beyond two servers to accommodate a very popular file).

Figure 4 illustrates ADAPTLOAD which, unlike the algorithm EQUILOAD of Figure 3, uses past workload information to determine (fuzzy) boundary points for the future. Also, since ADAPTLOAD is an online algorithm, it must manipulate DDHs efficiently (in linear time). Thus, instead of storing a DDH with a vector of size equal to the number of files, we use a vector with a constant number F of *bins*, so that, for $1 \le f \le F$, bin f accumulates the total number of bytes t_f due to requests for files with size between C^{f-1} and C^f, in the (portion of the) trace being considered[4]. Thus a DDH is now expressed simply as $\{(f, t_f) : 1 \le f \le F\}$. Accordingly, the (possibly fuzzy) boundaries are expressed in terms of bin indices, not actual file sizes.

[4] C is some real constant greater than one. Using a value close to one results in a fine DDH representation, but also in a larger value for F, since C^F must exceed the size of the largest file that may be requested.

4 Performance analysis of ADAPTLOAD

This section presents a detailed performance analysis of ADAPTLOAD via trace-driven simulation. We selected traces for two consecutive days, June 26 and 27, as representative (Figure 2). During these two days, 52 and 18 million requests were served, respectively. From each trace record we select two values, the request arrival time and the size of the requested file (in bytes).

As described in the previous section, ADAPTLOAD balances the load on the back-end servers using its knowledge of the past workload distribution. Specifically, the algorithm of Figure 4 schedules the i^{th} batch of K requests according to boundaries computed using the $(i-1)^{\text{th}}$ batch of K requests[5]. As expected, the performance of the policy is sensitive to the value of K. For the World Cup 1998 workload, we experimented with several values for K and $K = 50{,}000$ appears to be a good choice. Indeed, values between 25,000 and 75,000 results in almost indistinguishable performance, while values in the hundred of thousands are poor due to the high variability over time in the arrival process. In Section 4.3 we elaborate on alternative ways to use previous workload to predict the future one.

We compare ADAPTLOAD against the *Join Shortest Weighted Queue* (JSWQ) policy, where the length of each queue in the system is weighted by the size of queued requests[6]. We focus on the following questions:

Can ADAPTLOAD *respond quickly to transient overload?* To answer this question, we report performance metrics as a function of time, i.e., we plot the average slowdown perceived by the end user during each time interval

[5] For the first batch, previous history is not available. In our simulation we simply discard its corresponding performance data, so we use the first K requests just to compute the first DDH. In a practical implementation, instead, we would simply guess boundaries to be used for the first batch.

[6] We also experimented using the Join Shortest Queue (JSQ) policy but we do not report the results because they are consistently inferior to those for JSWQ.

1. set fuzzy boundaries $(s_1, 1), (s_2, 1), \ldots, (s_{N-1}, 1)$ to "reasonable guesses"
2. initialize the request counter: $R \leftarrow 0$, the total requested bytes: $B \leftarrow 0$, and the the DDH: $\forall f, 1 \leq f \leq F, t_f \leftarrow 0$
3. while $R < K$
  a. get new request, let its size be s, and increment R
  b. assign request to a server based on s and $(s_1, p_1), (s_2, p_2), \ldots, (s_{N-1}, p_{N-1})$
  c. insert request size into the DDH bin f such that $C^{f-1} \leq s < C^f$: $t_f \leftarrow t_f + s$
  d. increment B by the new request size: $B \leftarrow B + s$
4. initialize the server index $i \leftarrow 1$ and the accumulated weight $A \leftarrow 0$
5. for $f = 1$ to F scanning the DDH bins to update $(s_1, p_1), (s_2, p_2), \ldots, (s_{N-1}, p_{N-1})$:
  a. increment accumulated weight A with the weight of the f^{th} bin: $A \leftarrow A + t_f$
  b. while $A > B/N$
    I. set boundary s_i for server i to the current f: $s_i \leftarrow f$
    II. decrement A by B/N: $A \leftarrow A - B/N$
    III. set fraction p_i for server i to be: $p_i \leftarrow 1 - A/t_f$
    IV. increment server index i: $i \leftarrow i + 1$
6. go to **2.** and process the next batch of K requests

Figure 4. Setting the fuzzy boundaries $(s_1, p_1), (s_2, p_2), \ldots, (s_{N-1}, p_{N-1})$ with ADAPTLOAD.

corresponding to K requests. Since the system operates under transient overload conditions and is clearly not in steady state, our experiments focus on examining ADAPTLOAD's ability to respond to sudden bursts of arrivals and quickly serve as many requests as possible, as efficiently as possible.

What is the policy sensitivity to different hardware speeds? To address the common belief that replacing the servers with much faster ones solves the problem of transient overloads, we run our simulations assuming either "fast" or "slow servers". This is equivalent, in turn, to considering "low" and "high" load, respectively, and allows us to comment on ADAPTLOAD's ability to quickly flush backed-up queues.

Does the policy achieve equal utilization across servers? Since ADAPTLOAD bases its boundaries on knowledge of the workload distribution, we examine the per-server utilization as a function of time and comment on the policy's ability to distribute load effectively.

Does ADAPTLOAD *treat short jobs differently from long jobs?* This question refers to the policy's *fairness*. To measure the responsiveness of the system, we report the average request slowdown of the classes of requests defined by the request sizes intervals.

Does ADAPTLOAD *scale well with respect to the cluster size?* Since ADAPTLOAD's ability to balance the workload is a function of an effective mapping of different file sizes to specific servers, we explore the algorithm's scalability by running simulations using the same trace data but on an increased number of back-end servers.

Can we improve ADAPTLOAD*'s performance with smarter parameterization?* We address this issue by elaborating on alternative ways to use previous workload to predict the future. We introduce a new version of the algorithm based on a geometrically discounted history of the workload and report on its effectiveness.

4.1 Experimental results

For the first set of experiments we considered a cluster consisting of four back-end servers. Results are reported for JSWQ and ADAPTLOAD with K = 50,000. Figure 5 shows the average request slowdown in the cluster during the two-day trace under "low" and "high" load in the system. The average request slowdown closely follows the workload arrival intensities of Figure 2(a)-(b). During the "quiet" early morning to early afternoon hours, JSWQ does better than ADAPTLOAD, especially under low load. This is because the load in the system is so low that there is always an idle server for an incoming request. ADAPTLOAD, whose scheduling decisions are instead based exclusively on precomputed boundaries, may direct a request to a server that is already busy even when an idle server is available. During the periods of transient overload, however, we see a very different behavior: ADAPTLOAD greatly outperforms JSWQ and it consistently achieves lower average slowdowns (a log-scale is used for the vertical axis of the average request slowdown plots). During the most overloaded times, ADAPTLOAD is able to return faster to acceptable slowdown levels (see Figure 5(b)).

Figure 6 shows the utilization of each server across time for ADAPTLOAD and JSWQ, under high load and for the same two-days period as in Figure 5. JSWQ achieves a more uniform utilization across the four servers, while the per-server utilization of ADAPTLOAD has more variability,

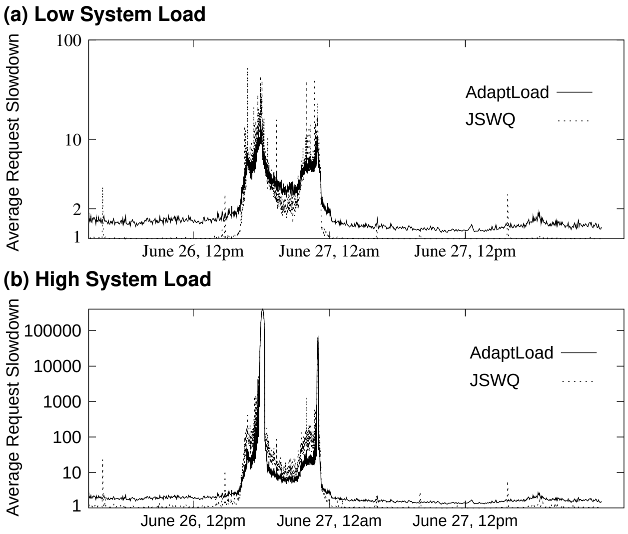

Figure 5. Average slowdown for ADAPTLOAD and JSWQ as a function of time, under either low or high load (four servers).

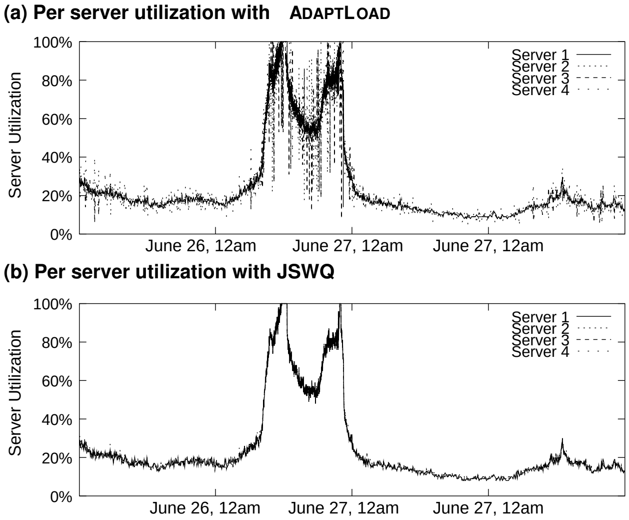

Figure 6. Server utilization with ADAPTLOAD or JSWQ as a function of time, under high load (four servers).

a direct effect of the algorithm's parameterization. Yet, even though the boundaries used are not optimal, ADAPTLOAD still achieves significantly better performance during overload periods.

Next, we consider the question of which requests are penalized most under ADAPTLOAD and JSWQ. Figure 7 illustrates the slowdown for various ranges of requests sizes for June 26, at 22:42:22, for the two policies, under either low or high load. The bar graph confirms that, with JSWQ, short requests (which constitute the bulk of the workload) are penalized most because they tend to get blocked in the queue behind large ones. This effect is apparent under either load, but is more pronounced under high load, as apparent from Figure 7(b). Instead, ADAPTLOAD manages to consistently maintain small slowdowns for nearly all classes of requests, improving the overall system performance.

4.2 Scalability

To examine ADAPTLOAD's scalability as a function of the number of back-end servers, we double the number of servers from four to eight. We focus on the policy performance using the trace data of June 27 alone, a day showing more uniform arrival intensity but also more variable service demands when compared with June 26.

Figure 8 shows the request slowdown with the ADAPTLOAD and JSWQ policies as a function of time, under either low or high load. In either case, JSWQ cannot process the requests effectively, resulting in an expected slowdown curve with more "spikes" than ADAPTLOAD. The system reaches saturation around 6 pm with either policy, but ADAPTLOAD recovers much faster than JSWQ. JSWQ is marginally better than ADAPTLOAD when the overall load is low, but substantially worse under overloads.

Figure 9(a) illustrates the request slowdown over the spectrum of request sizes with eight servers, for June 27 at 11:43:18, under low load. In this case, JSWQ performs slightly better than ADAPTLOAD, but note that the range of the y-axis is very narrow, from 0 to 2.5. This is a direct effect of the fact that the largest requests are quite rare, but our conservative estimation of boundaries allocates "too many" resources for the large requests, resulting in a reduction of the resources available to satisfy requests smaller than 1 MByte.

Figure 9(b) presents the per-class slowdown for June 27 at 20:37:32, under high load. Because of the presence of the whole range of requests in the K requests that correspond to the selected measurement point, ADAPTLOAD greatly outperforms JSWQ for all request ranges with the exception of the $[0.1-1]$ MByte range. Even for cases where the number of servers per cluster is large but only few popular files in the discrete data histogram contribute to the majority of the distribution, these few files are served by multiple servers, and ADAPTLOAD still maintains high performance.

4.3 Improving ADAPTLOAD

Recall that ADAPTLOAD's ability to balance the load is heavily influenced by the ability of predicting the distribution of the upcoming workload, based on knowledge about the past K requests. In this section, we propose an alternative approach that uses information about the entire past to increase policy responsiveness.

As with the basic ADAPTLOAD algorithm, we partition

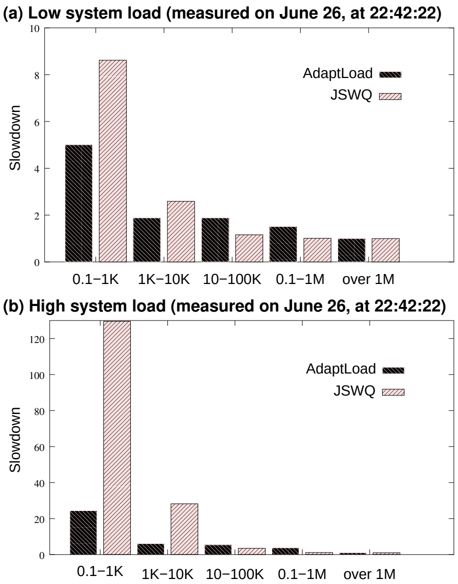

Figure 7. Slowdown by class of request sizes under either low or high load (four servers).

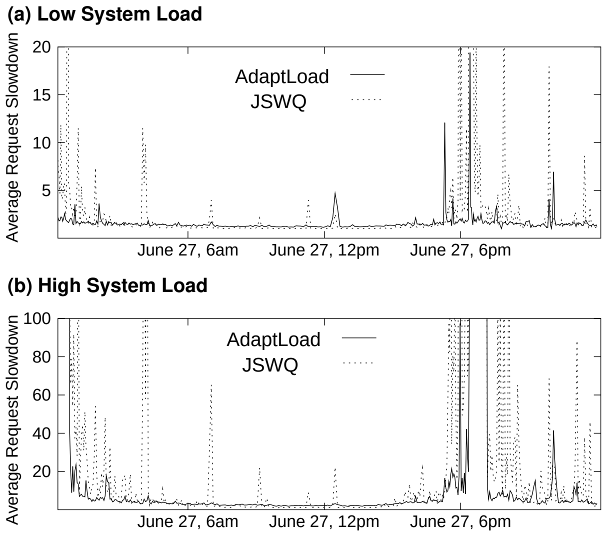

Figure 8. Average request slowdown of ADAPTLOAD and JSWQ policies under either low or high load (eight servers).

K	α for low load	α for high load
32768	0.2986328	0.0750000
8192	0.6998535	0.6228516
1024	0.9835938	0.9374023
512	0.9820313	0.9750000
256	0.9281250	0.9875000

Table 1. Optimal α values as a function of K, under high or low load.

the flow of requests into batches of K requests and use a DDH to recompute the boundaries of every batch. However, instead of considering only the last batch, we now use information about *all* batches seen so far. Let O_i be a vector representing the DDH *observed* in the i^{th} batch. Then, the DDH U_i *used* to allocate a batch is obtained as a *geometrically discounted* weighted sum of all the previously observed batches:

$$U_{i+1} = \frac{\sum_{j=0}^{i} \alpha^{i-j} O_j}{\sum_{j=0}^{i} \alpha^{i-j}} = \frac{(1-\alpha)O_i + (\alpha - \alpha^{i+1})U_i}{1-\alpha^{i+1}}.$$

Where the positive coefficient α, $0 \leq \alpha \leq 1$, controls the rate at which memory of the past decreases in importance (the case $\alpha = 0$ corresponds to the algorithm previously presented, where only the last batch i is used to compute U_{i+1}; the case $\alpha = 1$ corresponds to giving the same weight to all batches). Since this version of ADAPTLOAD takes into consideration the whole history of the workload, K can be smaller comparing to the basic version of the policy, thus allowing for faster adaptations to workload changes. For any given trace and value of K, it is possible to find an *a posteriori* value of α providing nearly optimal performance: obviously, as a trend, the larger K, the smaller α. The α values determine how much of the past should be used to predict the future load.

ADAPTLOAD proves not to be very sensitive to (K, α) pairs. Table 1 reports α values that result in the lowest maximum observed queue length, for various K values and under either low or high load. For the computation of K we used the same two days of the trace as in the previous experiments. These values were found using an exhaustive search. Our exhaustive search indicated that it is possible to find a range of α's providing nearly-optimal performance. Experimenting with a fixed $K = 512$ and values of within the range $[\alpha - 0.2, \alpha + 0.1]$ indicated that performance is very close to that with optimal α.

Finally, we turn to the performance improvements with geometrically discounted history. Figures 10 illustrates the expected slowdown as a function of time for two versions of ADAPTLOAD: the basic one that considers only the immediate previous history, and one that uses all prior history using the optimal value of α. The selected time period are the peak hours of June 26, and we explored both low and high load. In both cases, the version of ADAPTLOAD with geometrically discounted history performs much better.

5 Conclusions

We presented ADAPTLOAD, a new adaptive policy to allocate static requests to back-end processors in a clustered

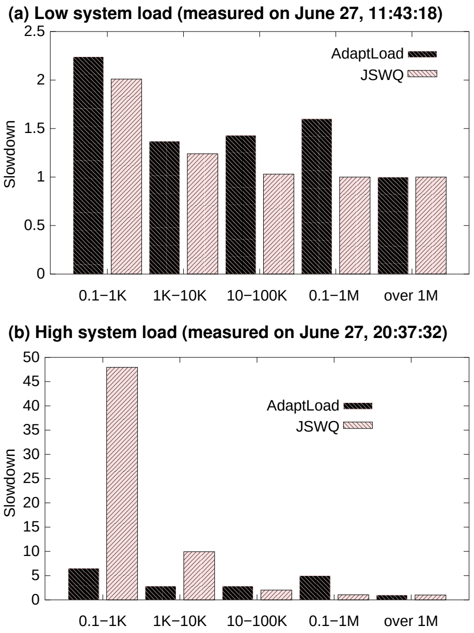

Figure 9. Slowdown by class of request sizes under either low or high load (eight servers).

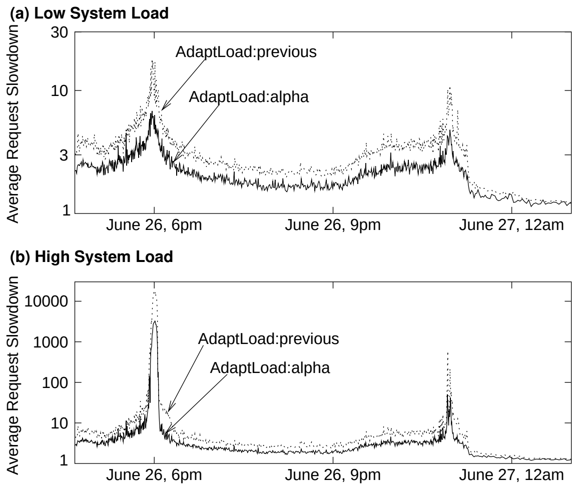

Figure 10. Average request slowdown for ADAPTLOAD **under either low or high load (eight servers) with ($K = 512$, $\alpha = 0.92$).**

web server, based on the size of the requested documents. With the goal to minimize the slowdown experienced by individual requests, the policy assigns a range of sizes to each processor, and makes its allocation decision based on the range containing the size of a given request.

As a good choice of the boundaries defining the ranges is paramount, we propose two dynamic techniques to decide these values, based on grouping the incoming requests into sequential batches. The first one estimates the request size distribution on one batch to decide the allocation boundaries for the next batch, and is shown to greatly outperform the successful Join-Shortest-Weighted-Queue policy when it matters most: during heavy load and periods of server saturation. The second one uses the entire history of requests in a geometrically discounted way and smaller batch sizes to increase responsiveness to workload fluctuations, and is shown to result in even lower slowdowns.

References

[1] M. Arlitt and T. Jin. Workload characterization of the 1998 World Cup Web Site. HP Labs Technical Report, Hewlett Packard, Palo Alto, CA, Sept. 1999.

[2] M. Arlitt, D. Krishnamurthy, and J. Rolia. Characterization and performance scalability of a large web-based shopping system. *ACM Trans. Internet Tech.*, 1(1):44–69, 2001.

[3] P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella. Changes in Web Client Access Patterns: Characteristics and Caching Implications. *World Wide Web*, 2 (Special Issue on Characterization and Performance Evaluation):15–28, 1999.

[4] P. Barford and M. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proc. 1998 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pages 151–160, Madison, WI, June 1998. ACM Press.

[5] V. Cardellini, M. Colajanni, and P. S. Yu. DNS dispatching algorithms with state estimators for scalable web-server clusters. *World Wide Web*, 2(3):101–113, July 1999.

[6] V. Cardellini, M. Colajanni, and P. S. Yu. Load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39, May/June 1999.

[7] G. Ciardo, A. Riska, and E. Smirni. EQUILOAD: a load balancing policy for clustered web servers. *Perf. Eval.*, 46(2-3):101–124, Oct. 2001.

[8] M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. In *Proc. 10th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Lecture Notes in Computer Science 1469, pages 231–242. Springer-Verlag, 1998.

[9] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proc. ACM 8th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, pages 205–216, San Jose, CA, Oct. 1998.

[10] A. Riska, E. Smirni, and G. Ciardo. Analytic modeling of load balancing policies for tasks with heavy-tailed distributions. In *Proc. Workshop on Software Performance Analysis (WOSP)*, pages 147–157, Ottawa, Canada, Sept. 2000. ACM Press.