Deferred Assignment Scheduling in Cluster-based Servers

Victoria Ungureanu * Benjamin Melamed † Michael Katehakis ‡
Phillip G. Bradford §

Abstract

This paper proposes a new scheduling policy for cluster-based servers called `DAS` (Deferred Assignment Scheduling). The main idea in `DAS` is to defer scheduling as much as possible, in order to make better use of the accumulated information on job sizes. In broad outline, `DAS` operates as follows: (1) incoming jobs are held by the dispatcher in a buffer; (2) the dispatcher monitors the number of jobs being processed by each server; (3) when the number of jobs at a server queue drops below a prescribed threshold, the dispatcher sends to it the *shortest job* in its buffer.

To gauge the efficacy of `DAS`, the paper presents simulation studies, using various data traces. The studies collected response times and slowdowns for two cluster configurations under multi-threaded and multi-process back-end server architectures. The experimental results show that in both architectures, `DAS` outperforms the `Round-Robin` policy in all traffic regimes, and the `JSQ` (Join Shortest Queue) policy in medium and heavy traffic regimes.

Keywords: clustered servers, deferred assignment, heavy-tail distribution, scheduling, simulation.

*Department of MSIS, Rutgers University, 180 University Ave., Newark, NJ 07102, email: ungurean@research.rutgers.edu

†Department of MSIS, Rutgers University, 94 Rockafeller Rd., Piscataway, NJ 08854, email: melamed@rbs.rutgers.edu

‡Department of MSIS, Rutgers University, 180 University Ave., Newark, NJ 07102, email: mnk@andromeda.rutgers.edu

§Department of Computer Science, The University of Alabama, Box 870290, Tuscaloosa, AL 35487-0290, email: pgb@cs.ua.edu

1 Introduction

Web servers are becoming increasingly critical as the Internet assumes an ever more central role in the telecommunications infrastructure. The proper operation of business-related applications (e.g., Web, multimedia and e-commerce activities, to name a few) depends on the efficient performance of Web servers. Applications that handle heavy loads, commonly use *cluster-based* architectures for Web servers, which combine good performance and low cost.

A cluster-based server consists of a front-end dispatcher and several back-end servers (see Figure 1). The dispatcher receives incoming requests and then decides how to assign them to back-end servers, which in turn serve the requests according to some discipline. The dispatcher is also responsible for passing incoming data pertaining to a job from a client to a back-end server; so, for each job in progress at a back-end server there is an open connection between the dispatcher and that server [17, 23].

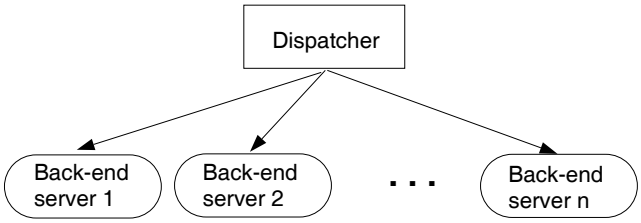

Figure 1: A cluster-based Web server.

A number of dispatcher assignment policies have been proposed for this type of architecture (see Section 5), but to the best of our knowledge, all these policies make the assumption that *the dispatcher*

sends requests to back-end processors immediately upon job arrival. In contrast, our approach calls for *deferred assignment* rather than immediate assignment. By deferred assignment we simply mean that the (front-end) dispatcher *defers* the assignment of jobs to (back-end) servers, rather than assign them immediately upon their arrival. The rationale for the deferred assignment approach is straightforward: Judicious assignment deferral enables the dispatcher to make better assignment decisions, resulting in greatly improved performance, especially in heavy traffic regimes.

To this end, we propose a policy, called `DAS` (Deferred Assignment Scheduling), which in broad outline operates as follows: (1) incoming jobs are held by the dispatcher in a buffer; (2) the dispatcher monitors the number of jobs being processed by each server; (3) when the number of jobs at a server queue drops below a prescribed threshold, the dispatcher sends to it the *shortest job* in its buffer. We argue that `DAS` has several important merits:

- good performance, especially under traffic surges
- amenability to practical implementation
- resistance to denial-of-service attacks

We now proceed to discuss the merits of `DAS` in some detail.

First, the good performance of `DAS` is supported by experimental results, using several data traces. The experimental studies collected response times and slowdowns under both multi-threaded and multi-process back-end server architectures. The experimental results show that in both architectures, `DAS` outperforms the `Round-Robin` policy in all traffic regimes, and the `JSQ` (Join Shortest Queue) policy under medium and heavy traffic regimes. The latter is a significant result, since it has been proven that `JSQ` has optimal response-time statistics when the dispatcher uses immediate assignment, and job arrival and sizes follow certain distributions [25] (we will revisit this result in Section 5).

Secondly, `DAS` is amenable to practical implementation in that the amount of information used by the dispatcher is relatively small, so that the attendant computation and communication overheads are low. More specifically, the dispatcher need only estimate the size of jobs and need only know when the number of jobs currently processed by a back-end server drops below a prescribed threshold. Indeed, job sizes in common Web applications can be easily estimated from the size of the document requested. Furthermore, the dispatcher can be explicitly notified by servers when their workload drops below the threshold, or alternatively, the dispatcher can infer this type of information by monitoring its number of active connections to back-end servers.

Finally, `DAS` can better resist denial-of-service attacks, because when the dispatcher identifies suspect arrival patterns, it can take countermeasures, such as discarding suspicious jobs. Since the dispatcher keeps and defers jobs, these measures can be taken before jobs are assigned to back-end servers, and thus *before malicious jobs consume back-end server resources.*

The rest of the paper is organized as follows. Section 2 discusses the `DAS` policy, and Section 3 introduces a motivational example. Section 4 presents comparative several performance studies for `DAS`, `Round-Robin` and `JSQ`, based on simulations driven by two data traces. Section 5 presents related work in the literature and Section 6 concludes the paper.

2 The DAS Policy

The `DAS` policy implements the *deferred assignment* principle, which is based on the premise that superior scheduling may be achieved by allowing the dispatcher to defer the assignment of incoming jobs to back-end servers. Under `DAS`, the dispatcher has a threshold parameter, θ, for deciding when to assign a job to a back-end server. The `DAS` policy, with threshold θ, is denoted by `DAS(`θ`)`, and has the following rules of operation.

1. *The dispatcher keeps incoming jobs in a buffer and monitors the number of requests processed by each server.*
2. *When the number of jobs processed by a back-end server,* `S`*, drops below* θ*, the dispatcher assigns to* `S` *the job with the shortest estimated size in its buffer (if the buffer is not empty).*
3. *If the dispatcher buffer is empty when a back-end server drops below*

θ, then that server is entered by the dispatcher in a queue. If the queue is not empty when a new job arrives, the dispatcher immediately assigns the job in question to the back-end server at the head of the queue.

We draw the reader's attention to the following points. First, the dispatcher estimates job sizes based on the *size of requested files*. Although *job service time* is a more relevant metric pro forma, we nevertheless chose *size*, because the dispatcher has this type of information readily available. Furthermore, it has been shown that the time to service a request is well-approximated by the size of the requested file [11].

Secondly, to prevent *starvation* of large jobs, the dispatcher periodically updates estimated sizes of jobs in its queue. In essence, the estimated size of a job decreases with the time it has waited in the dispatcher queue. Consequently, even large jobs will eventually appear to the dispatcher as having small sizes, and thus accelerate their selection for assignment. This feature is implemented as follows: the dispatcher records for each job the time it has last updated its size. If the time since last update exceeds a predefined interval $\mathtt{T_u}$, then the estimated size is divided by a factor $\mathtt{F_u}$. The values for $\mathtt{T_u}$ and $\mathtt{F_u}$ used in the experimental study (presented in Section 4) are respectively, 100 ms and 2. In effect, by using these values, the estimated size of a file is reduced by a factor of 1000 in 1s, and thus even very large files are not penalized too long.

Finally, we point out that the choice of the threshold value, θ, is a compromise reconciling two opposing goals. On one hand, one wishes to defer assignment as much as possible, thus affording the dispatcher the opportunity of making a better assignment decision by choosing from a larger pool of jobs. On the other hand, one should avoid keeping back-end servers idle while jobs are awaiting processing at the dispatcher. In summary, the first goal calls for a small threshold, and the second for a large one.

We next proceed to illustrate the efficacy of `DAS` by comparing its performance to various policies in two settings. First we present in Section 3 a brief motivational example, and then we show in Sections 4 the results of several experimental studies driven by two data traces.

3 A Motivational Example

In this example, we compare `DAS` with the following policies:

1. `Round-Robin`: Jobs are assigned to back-end servers $1, 2, \ldots, n$ in a cyclical manner, namely, the i-th task is assigned to server $i \bmod n + 1$. This policy equalizes the number of jobs assigned to each server.

2. `Size-Range`: Each host serves jobs whose service demands fall in a particular service-time range. This type of policy attempts to keep small jobs from getting stuck behind large ones. Examples of this type of policy include SITA-E [10], EquiLoad [4], and AdaptLoad [19].

3. `Join Shortest Queue (JSQ)`: Each incoming job is assigned to the back-end server with the smallest amount of residual work, i.e., the sum of service demands of all jobs in the server queue plus the residual work of the jobs currently being served.

In order to evaluate the relative efficacy of these policies, we compared their performance with respect to *response time*, defined as the time interval from the moment a request arrives at the dispatcher and up until it ends processing at the corresponding back-end server. Specifically, we computed the average response time yielded by `Round-Robin`, `JSQ`, `Size-Range` and `DAS`, for the sequence of jobs presented in Figure 2, served by a cluster of two back-end servers.

In summary, the performance results are as follows. `Round-Robin`, `JSQ` and `Size-Range`, which use immediate assignment, yield average response time in excess of 20 ms. In contrast, `DAS`, which uses deferred assignment, performs far better, yielding an average response time of only 5 ms! This exceptionally low average response time is attained, because the dispatcher defers the long job J_3, assigning with priority small jobs, which affords them the opportunity to be processed faster. In contrast, under `Round-Robin`, `JSQ` and `Size-Range`, job J_3 is assigned immediately, and consequently delays the processing of all short jobs arriving after it.

Job ID	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	...	J_{49}	J_{50}
Arrival time (ms)	1	2	3	4	5	6	7	8	...	49	50
Required service time (ms)	3	3	100	2	2	2	2	2	...	2	2

Figure 2: A motivational example.

We show now how the response time values are computed. The computation makes the following simplifying assumptions: (1) communication times between the dispatcher and back-end servers are negligible, (2) the overhead incurred by the dispatcher to select (job, server) pairs is negligible, and (3) jobs are processed in first come, first served order. (We point out that the simulation studies presented in Section 4 do not make this last assumption.)

If the dispatcher assigns the jobs in a `Round-Robin` manner, then the first back-end server (S_1) sequentially receives jobs: J_1, J_3, J_5, J_7,..., J_{49} at the arrival times above. Likewise, the second back-end server (S_2) receives jobs: J_2, J_4, J_6, J_8,..., J_{50} at the corresponding arrival times. Denoting by R_k^S the response time of job k at server S, the response times at server S_1 are $R_1^1 = 3$, $R_3^1 = 101$, $R_5^1 = 101$, $R_7^1 = 101$, etc. Similarly, at server S_2, $R_2^2 = 3$, $R_4^2 = 3$, $R_6^2 = 3$, etc. Thus, the average response time for this cluster is:

$$\frac{\sum_{j=0}^{24} R_{2j+1}^1 + \sum_{j=1}^{25} R_{2j}^2}{50} = \frac{3 + \sum_{j=1}^{24} 101 + \sum_{j=1}^{25} 3}{50} \approx 46.$$

The poor performance of the `Round-Robin` policy is due to job J_3, which requires a service time of 100, and is scheduled by server S_1 before the smaller jobs J_5, ..., J_{49}.

Now consider the case where the dispatcher uses a `Size-Range` policy for assigning requests to back-end servers. Assume further that server S_1 is assigned jobs requiring service times in excess of 10 time units each, while server S_2 is assigned the smaller jobs. This policy gives rise to the following assignments: server S_1 is assigned job J_3, with all other jobs assigned to server S_2. Notice that the load is evenly distributed between the two servers, each receiving jobs that require approximately 100 time units of service. Under this policy, $R_3^1 = 100$, $R_1^2 = 3$, $R_2^2 = 5$, $R_4^2 = 5$, $R_5^2 = 6$, $R_6^2 = 7$, etc. The average response time is:

$$\frac{R_3^1 + R_1^2 + R_2^2 + \sum_{j=4}^{50} R_j^2}{50} = \frac{100 + 3 + 5 + \sum_{j=4}^{50} j + 1}{50} \approx 25.$$

Here, the long job (J_3) and the set of short jobs are assigned to different servers. However, the average response time is still large, because server S_2 cannot process the short jobs as fast as they arrive, and so the short jobs must wait longer and longer to be served.

An average response time of comparable magnitude is also obtained when the dispatcher uses the `JSQ` policy. In this case, server S_1 is assigned jobs J_1 and J_3, while server S_2 is assigned the rest of the jobs. Under this policy, the response times at server S_1 are $R_1^1 = 3$ and $R_3^1 = 101$; and the response times at server S_2 are $R_2^2 = 3$, $R_4^2 = 3$, $R_5^2 = 4$, $R_6^2 = 5$, $R_7^2 = 6$, etc. The corresponding average response time is:

$$\frac{R_1^1 + R_3^1 + R_2^2 + \sum_{j=4}^{50} R_j^2}{50} = \frac{2 + 101 + 3\sum_{j=4}^{50} j - 1}{50} \approx 22.$$

Finally, consider a dispatcher that uses the `DAS` policy. For simplicity we considered here the case when θ=1, meaning that the dispatcher assigns a job to a server only when the server becomes idle. In this case, server S_1 would be assigned jobs J_1, J_4, J_6, J_8, ...,J_{50}, while S_2 would be assigned jobs J_2, J_5, J_7,..., J_{49} and J_3, *in that order*. In this schedule, every short job is served without delay, whereas the long job is served last, yielding an average response time of:

$$\frac{R_1^1 + \sum_{j=2}^{25} R_{2j}^2 + R_2^2 + \sum_{j=2}^{24} R_{2j+1}^2 + R_3^2}{50} =$$

$$\frac{3 + \sum_{j=2}^{25} 2 + 3 + \sum_{j=2}^{24} 2 + 148}{50} \approx 5.$$

This exceptionally low average response time is attained, because whenever a server becomes idle, the dispatcher has a small job on hand to assign to it. Consequently, the deferred assignment of the long job affords short jobs the opportunity to be served immediately.

4 Experimental Performance Studies

Recall that the rationale underlying the `DAS` policy is the assumption that deferring job assignment enables the dispatcher to make better assignment decisions. A case in point is our motivational example, where `DAS` yielded significantly better average response times than policies that use immediate assignment.

We next show that this assumption comes true for simulation experiments with various cluster architectures, driven by a variety of data traces. The experiments were subject to the assumptions 1 and 2, made in Section 3, namely, that communication times between the dispatcher and back-end servers are negligible, and the overhead incurred by the dispatcher to select (job, server) pairs is negligible as well.

The `DAS` policy used was `DAS(2)` (i.e., a threshold value of 2, so that the dispatcher assigns a request to a back-end server only when the number of jobs there drops below 2). This ensures that back-end servers are never idle when there are jobs awaiting processing. The immediate assignment policies compared with `DAS(2)` were `Round-Robin` and `JSQ`. We excluded from the study `Size-Range` policies, because it was shown in [19] that they are outperformed by `JSQ` for the traces used in our study.

We compared these assignment policies via two performance metrics: response time and slowdown (the ratio between a job's response time and its service time). More specifically, the following statistics were used:

- **Average response times in successive time intervals.** The time horizon was divided into successive time intervals of fixed length, and the average response time in each such interval was computed.

- **Slowdown tail probabilities.** Tail probabilities of a random variable X have the form $T_X(t) = \mathbb{P}\{X > t\}$ (for large t), where the function $T_X(t)$ is the complementary cumulative distribution function of X. This statistic was selected, because QoS (Quality of Service) metrics are often stated in the form $\mathbb{P}\{S > t^*\} \leq \epsilon^*$, where S denotes the slowdown, and t^* and ϵ^* are prescribed values. Operationally, such a QoS requires that only a (small) fraction, ϵ^*, of jobs will experience a (large) slowdown exceeding an "acceptable upper bound", t^*. We estimate the tail probabilities of the slowdown, S, in the standard way, by the complementary cumulative relative frequencies $\hat{T}_S(t) = n_t/N$, where N is the total number of jobs serviced, and n_t is the number of jobs whose slowdown exceeds t.

We experimented with the following two back-end server architectures:

- **MT (Multi-Threaded).** In this architecture, a back-end server spawns multiple threads within a single address space. A thread serves a request to completion before accepting a new one.

- **MP (Multi-Process).** In this architecture a back-end server employs multiple processes. A process serves a request to completion before accepting a new one. Each process runs for a predefined time interval (quantum) or until it blocks, after which the operating system (OS) selects another process and runs it.

To contrast the the performance of assignment policies under these two architectures, we assumed non-preemptive (coroutine) scheduling for threads, where a thread is allowed to run until it finishes or blocks [6]. We excluded from consideration preemptive scheduling, since this would render threads and processes operationally indistinguishable. We mention that Apache, the most popular Web server running today [1, 12], uses the MP architecture over UNIX, and the MT architecture over Microsoft Windows NT.

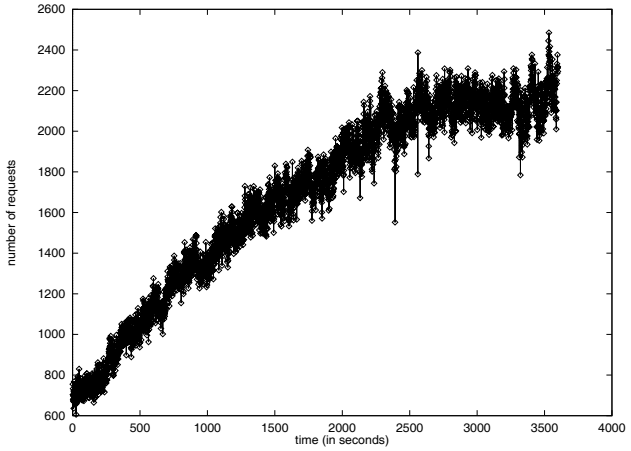

Figure 3: Number of request arrivals per second.

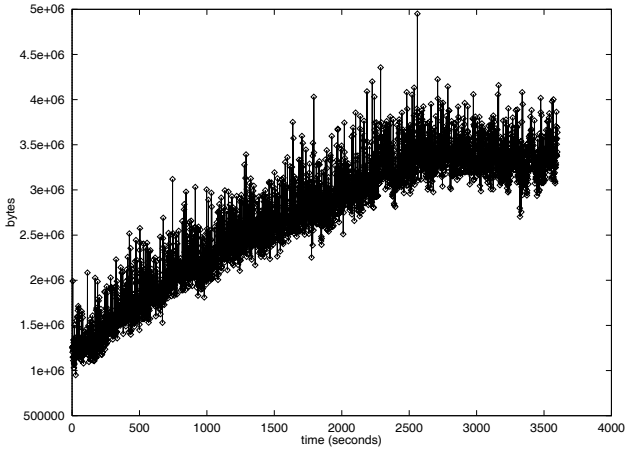

Figure 4: Total bytes requested per second.

The simulation results under the two aforementioned architectures will be presented in Sections 4.1 and 4.2 below.

4.1 Multi-Threaded Back-End Servers

This study used trace data from Internet sites serving the 1998 World Cup. The data used are available on the Internet Traffic Archive (see [2] and http://ita.ee.lbl.gov/html/traces.html). This repository provides detailed information about the 1.3 billion requests received by World-Cup sites over 92 days – from April 26, 1998 to July 26, 1998. We mention that Arlitt and Jin [2] have shown that job sizes from these traces follow a *heavy-tail distribution*, where a relatively small fraction of jobs accounts for a relatively large fraction of the overall load. From this repository, we selected a trace covering over 1 hour from the June 26 data, containing approximately 6 million requests. The relevant statistics of this trace are described next.

4.1.1 Simulation Data

Figure 3 depicts the number of requests received by the World-Cup cluster in successive one-second intervals, while Figure 4 plots the number of bytes requested from the same cluster in the same time intervals. Figure 5 is essentially a histogram depicting the logarithm of the number of requests made for various file sizes in the trace considered.

From each request record in the trace, we extracted only the request arrival time and the size of the requested file. The observed arrival rates were on the order of several hundreds of requests per second, and the recorded time stamps were rounded off to integer seconds. Consequently, we distributed request arrivals uniformly over each second. Since no service time information was recorded, the simulation estimated the service time as the sum of the (constant) time to establish and close a connection, and the (variable, size-dependent) time required to retrieve and transfer a file. The justification for this estimation method may be found in [17, 19, 23].

The selection of the particular data trace was motivated by the fact that it exhibits arrival-rate fluctuations corresponding to light, medium and heavy loadings in this temporal order, as evidenced by Figures 3 and 4. More specifically, the trace allows us to study the performance of assignment policies under various loading conditions, as follows:

- **Light loading.** In the time interval [0, 1500], the arrival rate is relatively low (below 1600 requests/sec), and the resultant utilization coefficient is relatively low ($\approx 40\%$).

- **Medium loading.** In the time interval (1500, 2000], the arrival rate is between 1600 and 2000 requests/sec, and the resultant utilization coefficient is intermediate ($\approx 50\%$).

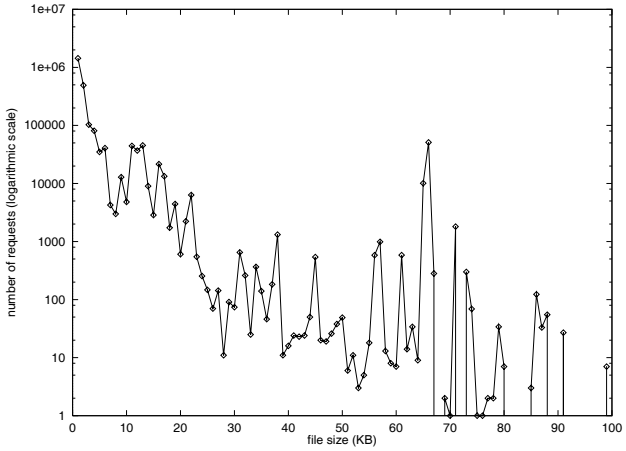

Figure 5: File size distribution of arriving requests (notice the logarithmic scale on the y axis)

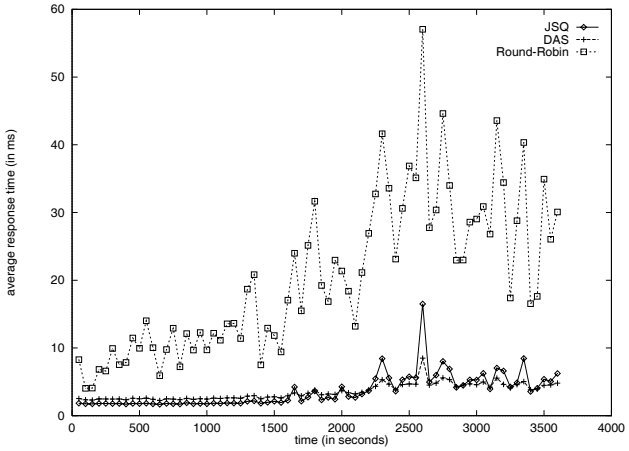

Figure 6: Average response time of `Round-Robin`, `JSQ` and `DAS` as function of time in an MT-cluster.

- **Heavy loading.** In the time interval (2000, 3600], the arrival rate exceeds 2000 requests/sec, and the corresponding utilization coefficient is relatively high (≈75%). In order to study the behavior of the slowdown under a heavy load regime, slowdowns were computed only for the jobs arriving in the heavily-loaded portion of the trace ([2000, 3600]).

4.1.2 Simulation Experiments

To compare the performance of various assignment policies, we simulated a cluster of four back-end servers, driven by the World Cup trace above.

Figure 6 displays average response times in successive 50-second intervals for the three assignment policies considered, under the MT architecture. Note that loading conditions vary over these successive intervals as per Figures 3 and 4. Figure 6 shows that `DAS(2)` improves over `Round-Robin` in all time intervals (and therefore, under all simulated loading conditions) by as much as one order of magnitude.

We next proceed to compare the average response times of `DAS(2)` to those of `JSQ`.

- **Light loading.** `JSQ` outperforms `DAS(2)` in the interval [0, 1500]. This behavior is explained by the fact that when the arrival rate is low, both `JSQ` and `DAS` effectively perform immediate assignment (more precisely, `DAS` defers jobs infrequently, since back-end servers are underloaded). In effect, the dispatcher information used by `DAS` is of little value. In contrast, `JSQ`, which attempts to balance load among back-end servers, uses information from back-end servers, which proves to be more relevant in this case. To sum up, under light loading, load-balancing outperforms deferred assignment.

- **Medium loading.** As the arrival rate and subsequent utilization increase in the interval (1500, 2000], the performance of `DAS(2)` is comparable to that of `JSQ`.

- **Heavy loading.** `DAS(2)` outperforms `JSQ` in the interval (2000, 3600] by as much as a factor of two. This behavior is explained by the fact that under `DAS`, more jobs wait in the dispatcher buffer, thereby precipitating more deferred assignments. To sum up, under heavy loading, deferred assignment outperforms load-balancing.

Figure 7 plots the complementary cumulative relative frequencies (tail probabilities) of slowdown for the three assignment policies considered, under the MT architecture. Recall that the lower the value of tail probabilities, the better the performance. We observe that `Round-Robin` again performed far worse than the other two policies. For example, under `Round-Robin` the probability that

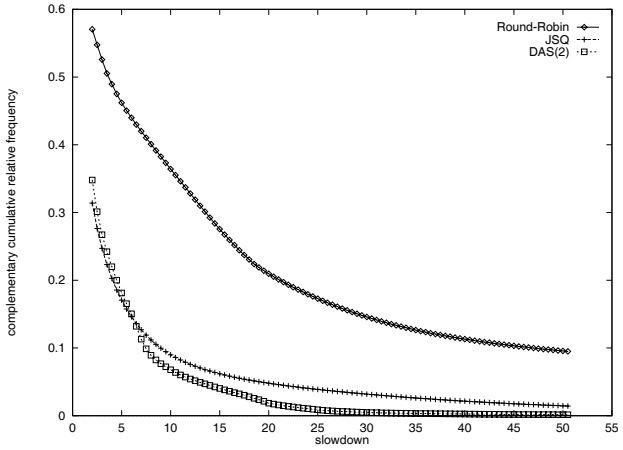

Figure 7: Complementary cumulative relative frequencies of slowdowns for `Round-Robin`, `JSQ` and `DAS` in an MT-cluster.

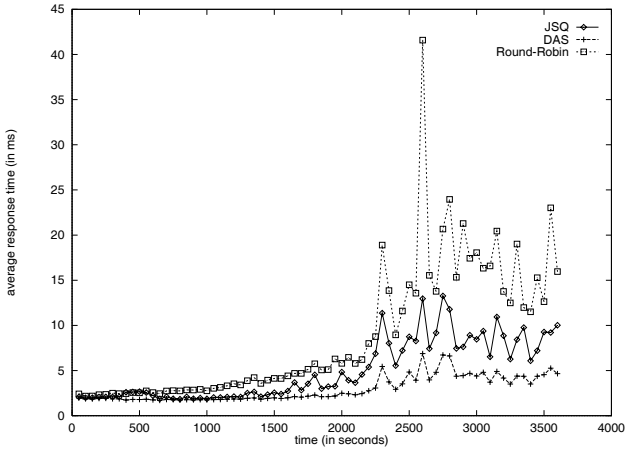

Figure 8: Average response time of `Round-Robin`, `JSQ` and `DAS` as function of time for an MP-cluster of four back-end servers.

a job has a slowdown in excess of 7 is 0.42, whereas under `JSQ` and `DAS(2)`, it is less than 0.12. We also note that `DAS(2)` performs similarly to `JSQ` for slowdowns less than 7, and it outperforms `JSQ` for slowdowns in excess of 7, albeit slightly.

4.2 Multi-Process Back-End Servers

For the MP architecture, the simulations used two additional parameters: process quantum size (set at 5 ms), and context switch time (set at 0.5 ms). Two configurations were considered under this architecture: a cluster of four back-end servers, and a cluster of eight back-end servers. The results of the corresponding studies are presented in Sections 4.2.1 and 4.2.2.

4.2.1 Cluster of Four Back-End Servers

This study used the World Cup data trace described in Section 4.1.1. Figure 8 displays average response times in successive 50-second intervals for an MP-cluster under the parameter settings above. Interestingly, under the MP-architecture, `DAS(2)` outperforms both `Round-Robin` and `JSQ` over *all* time intervals. The fact that `DAS(2)` outperforms `JSQ` even under light loading may be in part attributed to the (relatively expensive) context switching overhead, which increases the CPU load on the cluster. This leads to higher dispatcher buffer occupancies in `DAS`, allowing it to make more efficacious assignment decisions. In a similar vein, the relative advantage of `DAS(2)` over `Round-Robin` and `JSQ` increases as the load increases, analogously to the MT-case, though more strikingly. For heavy loading, `DAS(2)` outperforms `JSQ` by at least a factor of three.

Figure 9 plots the complementary cumulative relative frequencies (tail probabilities) of slowdown for the three assignment policies considered, under the MP architecture. We observe that `DAS(2)` consistently outperforms both `Round-Robin` and `JSQ` over *all* slowdown ranges. Furthermore, `DAS(2)` outperformed `JSQ` by a considerably wider margin than under the MT architecture (as compared to Figure 7). For example, the probability that a job has a slowdown in excess of 5 is 0.1 under `DAS(2)`, and 0.3 under `JSQ`.

4.2.2 Cluster of Eight Back-End Servers

This study gauged the performance of `JSQ` and `DAS(2)` as the cluster size grows larger. To this end, we generated a synthetic trace using a traffic generator, called Geist, designed at Intel [13, 14]. The relevant statistics of this trace are described next.

Simulation Data. The trace consists of over 1 million requests received in a time interval of

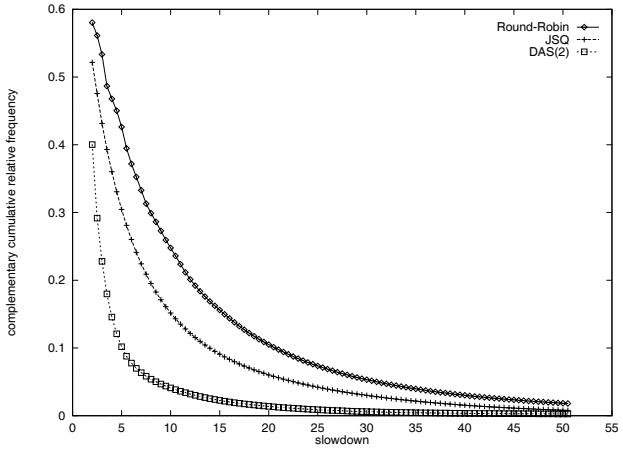

Figure 9: Complementary cumulative relative frequencies of slowdowns for `Round-Robin`, `JSQ` and `DAS` in an MP-cluster of four back-end servers .

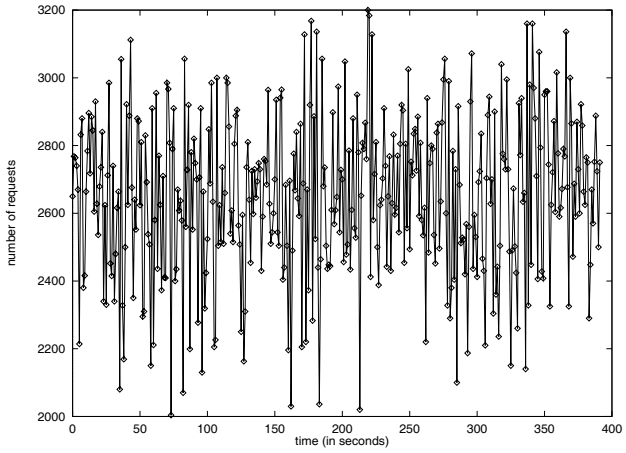

Figure 10: Number of request arrivals per second.

approximately five minutes. This trace offered the cluster medium loading with utilization in the range 45% to 55%. Figure 10 depicts the number of requests received by the cluster in successive one-second intervals, while Figure 11 plots the number of bytes requested from the same cluster in successive one-second intervals.

Simulation Experiments. This experiment compared the performance of `JSQ` and `DAS(2)` (the performance of `Round-Robin` is not depicted because it was substantially worse than both `JSQ` and `DAS(2)`). Figure 12 displays average response times

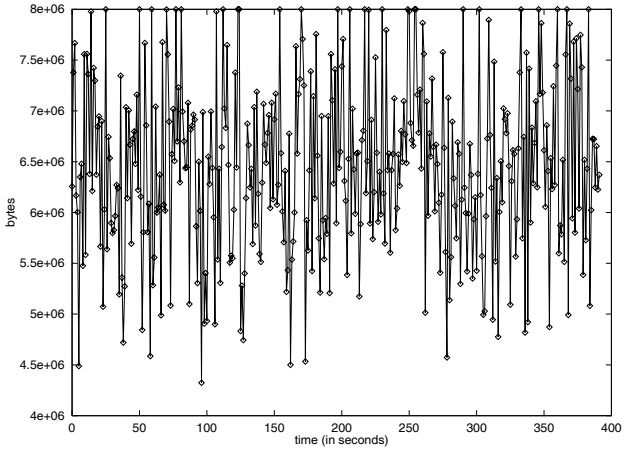

Figure 11: Total bytes requested per second.

in successive 10-second intervals for an MP-cluster of eight back-end servers. Observe that `DAS(2)` outperforms `JSQ` in most time intervals. Moreover, the performance advantage of DAS(2) over `JSQ` in a cluster of eight back-end servers is comparable to that in a cluster of four back-end servers.

Figure 13 plots the complementary cumulative relative frequencies (tail probabilities) of slowdown for the two assignment policies considered, for a cluster of eight MP back-end servers. Observe that `DAS(2)` consistently outperforms `JSQ` for slowdowns exceeding 3. Moreover, for slowdowns exceeding 7, `DAS(2)` outperforms `JSQ` by at least a factor of two.

4.3 Discussion

The simulation experiments show that `DAS` outperforms both `Round-Robin` and `JSQ` under medium and heavy loadings in terms of response time averages and tail probabilities, and this result holds across architectures and configurations. Moreover, it turns out that `DAS(2)` also outperforms `AdaptLoad` (an instance of a `Size-Range` policy) under medium and heavy loading offered by the World Cup trace. This follows from the fact that `JSQ` outperforms `AdaptLoad` under this particular trace [19].

The fact that `DAS` outperforms `JSQ` is all the more remarkable given the difference in the information used by the two policies. On one hand, `DAS` requires the dispatcher to estimate the sizes of in-

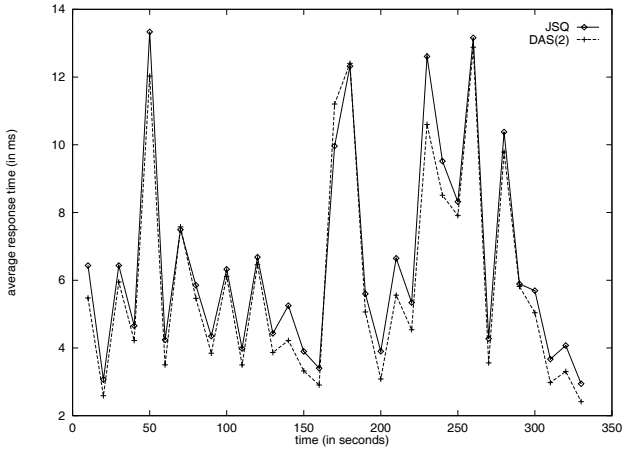

Figure 12: Average response time of JSQ and DAS as function of time for an MP-cluster of eight back-end servers.

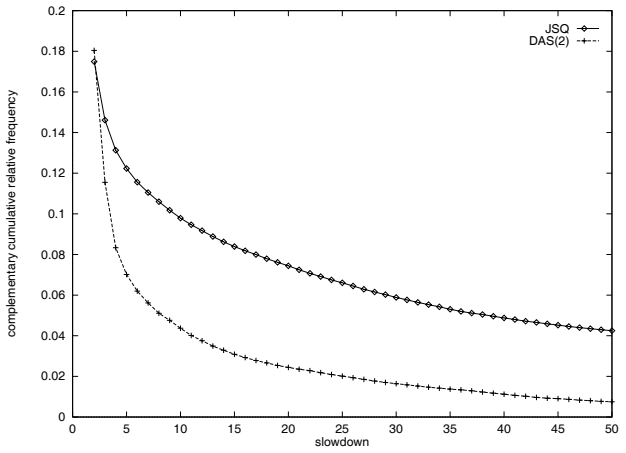

Figure 13: Complementary cumulative relative frequencies of slowdowns for JSQ and DAS in an MP-cluster of eight back-end servers.

coming jobs, and to monitor the *number of jobs* processed by back-end servers (recall that this information is readily available to the dispatcher). On the other hand, JSQ requires the dispatcher to track the *amount of residual work* at each server, which is much harder to estimate. For this reason, JSQ has been rarely, if at all, used in practice (cf. [17, 23]).

These results support the contention that DAS is a practical assignment policy alternative for cluster computing.

5 Related Work

The literature contains a considerable body of work on job scheduling (see [3, 5, 9, 15, 16, 18, 20] and references therein). We will briefly review here only size-based assignment policies. Smith [22] considered scheduling with fixed-size (deterministic) jobs on a single server. The paper shows that in this case, scheduling the shortest jobs first is optimal in that the policy yields minimal response times. In a similar vein, Rothkopf [21] shows that this policy yields minimal expected response times, for job sizes having arbitrary (known) distributions. Winston [25] considered a cluster-based server with the first-come first-serve (FCFS) discipline at each server queue, exponential job-size distributions, and Poisson arrivals. The paper proves that under these assumptions, the join-the-shortest-queue (JSQ) policy is optimal (yields minimal expected response times). However, Whitt [24] showed that there exist other job-size distributions for which JSQ is not optimal.

Next we proceed with a review of Size-Range scheduling policies that consider job sizes that follow a heavy-tail distribution. Harchol-Balter *et al.* [10] introduce a policy called "Size Interval Task Assignment with Equal Load" (SITA-E). The SITA-E policy fits job-size ranges (intervals) to bounded-Pareto distributions, and then equalizes the expected work. That is, given n back-end servers, then n size ranges are determined off-line, such that each range contains approximately the same amount of work. Ciardo *et al.* [4] presents a load-balancing policy, called EquiLoad, similar to SITE-E in that it also uses predefined ranges. The paper shows that EquiLoad performs well on World Cup data traces. The main drawback of SITA-E and EquiLoad is that they assume a priori knowledge of the job-size distribution. Another policy, called AdaptLoad, is proposed in [19] as an adaptive, on-line version of EquiLoad. Again, AdaptLoad assigns each back-end server to a job-size range, but these ranges are continually re-evaluated based on the most recent history window of requested jobs.

Finally, Harchol-Balter *et al.* [11] use the SRPT (Shortest Remaining Processing Time) when scheduling for execution processes at a back-end server. (Under SRPT, the OS of a back-end server

gives priority to processes serving short jobs or with short remaining processing time.) [11] shows experimentally that SRPT yields better response time performance than the traditional Round-Robin scheduling of processes. It is worth pointing out that both DAS and SRPT give priority to short jobs in order to improve response time performance. However, there are several major differences in the way they operate. First, DAS is employed by the dispatcher to assign jobs to back-end servers, whereas SRPT is employed by the OS of back-end servers to schedule the processing of jobs already assigned to them. Consequently, SRPT is more difficult to implement than DAS, because the former requires operating system modification, whereas the latter requires only the modification of the dispatcher application. Secondly, DAS, unlike SRPT, guarantees the long jobs do not starve. Finally, DAS provides a degree of resistance to denial-of-service attacks, while SRPT does not.

6 Conclusion

In this paper, we propose a novel approach to the job assignment problem, whereby the dispatcher is not forced to assign jobs to back-end servers upon request arrival; rather, the dispatcher may defer assignment by waiting to accumulate more information to greater advantage. Indeed, we have shown experimentally that this approach results in excellent response-time and slowdown performance as compared to traditional approaches.

These results support our contention that deferral-based assignment is very effective during traffic surges. For the duration of a surge, DAS gives priority to (many) small jobs, while the considerably less frequent large jobs are temporarily deferred. The fact that small jobs substantially outnumber large ones is backed up by empirical evidence suggesting that the sizes of files traveling on the Internet are heavy-tailed [7, 2, 8]. Indeed, for the traces considered, files with sizes greater than 30KB make up less than 3% of the files requested, but account for over 50% of the transferred data.

A consequence of these results is that a cluster using DAS is relatively resistant to denial-of-service attacks, which induce an artificially heavy loading. This is so, because DAS performance excels under heavy loading. Moreover, because most incoming jobs are deferred under heavy loading, the dispatcher has the opportunity to examine and discard malicious jobs before they consume back-end server resources.

References

[1] The Apache HTTP Server Project. http://httpd.apache.org/

[2] Arlitt, M. and T. Jin. Workload Characterization of the 1998 World Cup Web Site. *IEEE Network*, 14(3):30-37, May/June 2000. Extended version: Tech Report HPL-1999-35R1, Hewlett-Packard Laboratories, September 1999.

[3] Bruckner, P. *Scheduling Algorithms*, Third Edition, Springer-Verlag, 2001.

[4] Ciardo, G., A. Riska and E. Smirni. EquiLoad: A Load Balancing Policy for Clustered Web Servers. *Performance Evaluation*, 46(2-3):101-124, 2001.

[5] Colajanni, M., P. S. Yu and D. M. Dias. Analysis of Task Assignment Policies in Scalable Distributed Web-Server Systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(6), 1998.

[6] Coulouris, G., J. Dollimore and T. Kindberg. *Distributed Systems - Concepts and Design*. Addison-Wesley, 2001.

[7] Crovella, M.E., M.S. Taqqu and A. Bestavros. Heavy-tailed Probability Distributions in the World Wide Web. In *A Practical Guide To Heavy Tails*, Chapman Hall, New York, pages 3–26, 1998.

[8] Faloutsos, M., P. Faloutsos and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proceedings of ACM SIGCOMM '99*, pages 251-262, Aug. 1999.

[9] Harchol-Balter, M. Task Assignment with Unknown Duration. *Journal of the ACM*, 49(2):260-288, March 2002. (Extended Abstract in *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS)*, Taipei, Taiwan, April 2000.)

[10] Harchol-Balter M., M.E. Crovella and C.D. Murta. On Choosing a Task Assignment Policy for a Distributed Server System. In *Proceedings of Performance Tools '98*, Lecture Notes in Computer Science, 1468:231-242, 1998.

[11] Harchol-Balter M., B. Schroeder, N. Bansal, M. Agrawal. Size-based Scheduling to Improve Web Performance. *ACM Transactions on Computer Systems*, 21(2), May 2003.

[12] Hu, Y., A. Nanda and Q. Yang. Measurement, Analysis and Performance Improvement of the Apache Web Server. *International Journal of Computers and Their Applications*, 8(4):217-231, 2001.

[13] Kant, K., V. Tewari, and R. Iyer. Geist: A generator of E-commerce and Internet Server Traffic. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 49-56, November 2001.

[14] Kant, K., V. Tewari and R. Iye. Geist: A Web Traffic Generation Tool Source. In *Proceedings of the 12th International Conference on Computer Performance Evaluation, Modeling Techniques and Tools*, Lecture Notes In Computer Science, 2324:227-232, 2002.

[15] Katehakis, M. and C. Melolidakis. On The Optimal Maintenance of Systems and Control of Arrivals in Queues. *Stochastic Analysis and Applications*, 8(2):12-25, 1994.

[16] Pinedo, M. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 2002.

[17] Pai, V.S., M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel and E. Nahum. Locality-aware Request Distribution in Cluster-based Network Servers. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, 1998.

[18] Righter, R. Scheduling in Multiclass Networks with Deterministic Service Times. *Queuing Systems* 41(4):305-319, 2002.

[19] Riska, A., W. Sun, E. Smirni and G. Ciardo. AdaptLoad: Effective Balancing in Clustered Web Servers Under Transient Load Conditions. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.

[20] Ross, S.M. *Probability Models for Computer Science*. Academic Press, 2002.

[21] Rothkopf, M.H. Scheduling with Random Service Times. *Management Science*, 12:703-713, 1966.

[22] Smith, W.E. Various Optimizers for Single-Stage Production. *Naval Research Logistics Quarterly*, 3:59-66, 1956.

[23] Teo Y.M. and R. Ayani. Comparison of Load Balancing Strategies on Cluster-based Web Servers. *Simulation, The Journal of the Society for Modeling and Simulation International*, 77(5-6):185-195, November-December 2001.

[24] Whitt, W. Deciding which Queue to Join: Some Counter Examples. *Operations Research*, 34(1):55-62, 1986.

[25] Winston, W. Optimality of the Shortest Line Discipline. *Journal of Applied Probability*, 14:181–189, 1977.

Victoria Ungureanu (ACM) is an assistant professor in the MSIS department at Rutgers University. She has a Ph.D. in Computer Science from Rutgers University.

Benjamin Melamed is a Professor II at the Rutgers Business School- Newark and New Brunswick, Department of MSIS. Melamed received a B.Sc. degree in Mathematics and Statistics from Tel Aviv University in 1972, and a M.S. and Ph.D. degrees in Computer Science from the University of Michigan in 1973 and 1976, respectively. He was awarded an AT&T Fellow in 1988 and an IEEE Fellow in 1994. He became an IFIP WG7.3 member in 1997, and was elected to Beta Gamma Sigma in 1998.

Michael N. Katehakis is Professor of Management Science in the Department of Management Science and Information Systems, at Rutgers. He studied at the University of Athens, Diploma (1974) in Mathematics, at the University of South Florida, M.A. (1978) in Statistics, and at Columbia University, Ph.D. (1980) in Operations Research. He won the 1992 Wolfowitz Prize (with Govindarajulu Z.)

Phillip G. Bradford (ACM) is on the faculty in Computer Science Department at the University of Alabama. He earned his Ph.D. at Indiana University in Bloomington, his MS at The University of Kansas and his BS at Rutgers University.