DIMACS Technical Report 2002-41
October 2002

Deferred Assignment Scheduling in Clustered Web Servers

by

Victoria Ungureanu[1]
Department of MSIS
Rutgers Business School - Newark and New Brunswick
Rutgers University
180 University Ave., Newark, NJ 07102
email:ungurean@rbs.rutgers.edu

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

Michael Katehakis Department of MSIS
Rutgers Business School - Newark and New Brunswick
Rutgers University
180 University Ave., Newark, NJ 07102
email: mnk@andromeda.rutgers.edu

Benjamin Melamed
Department of MSIS
Rutgers Business School - Newark and New Brunswick
Rutgers University
94 Rockafeller Rd., Piscataway, NJ 08854
email: melamed@rbs.rutgers.edu

[1]Work supported in part by DIMACS under contract STC-91-19999, and Information Technology and Electronic Commerce Clinic, Rutgers University

---

ABSTRACT

This paper proposes new scheduling policies for clustered servers, which are based on job size. The proposed algorithms are shown to be efficient, simple and easy to implement. They differ from traditional methods in the way jobs are assigned to back-end servers. The main idea is to defer scheduling as much as possible in order to make better use of the accumulated information on job sizes. Furthermore, the proposed algorithms are designed to work effectively with the class of job-size distributions often encountered on the Internet. To gauge the efficacy of the proposed algorithms, the paper presents an empirical case study that shows these algorithms perform well on input from real-life trace data measured at Internet clustered servers.

1 Introduction

Web servers are becoming increasingly critical, as the Internet assumes an ever more central role in the communications infrastructure. common business applications/services (e.g., E-commerce, Web multi-media, to name a few) depends on the efficient performance of Web servers. Furthermore, from a customer viewpoint, a key Quality-of-Service (QoS) performance metric of the service rendered by a Web site is its *response time*. To improve service response times, it is essential to understand issues, such as server architecture, and Internet traffic loads.

In this paper, we consider a clustered architecture for Web servers, as depicted in Figure 1. A *clustered server* consists of a front-end dispatcher and several back-end servers; in particular, a *clustered Web server* responds to HTML requests. The dispatcher receives incoming requests, and then decides how to allocate them to the back-end servers, which then serve the requests according to some discipline. These activities are collectively known as *scheduling*. Here, we assume that both job sizes and arrival times are random, and that the processing time of a job is proportional to the associated file size. Our goal is to devise efficient algorithms for job scheduling, such that job response time is kept low.

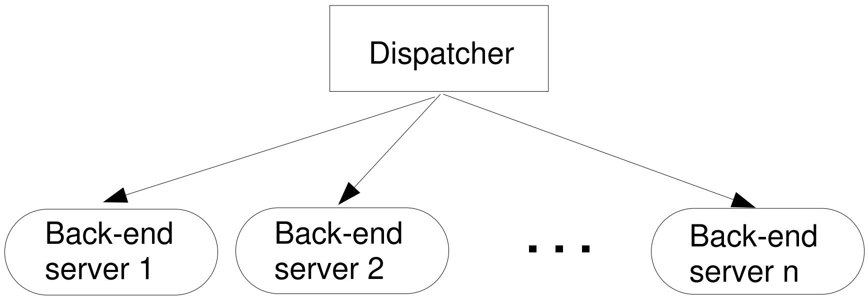

Figure 1: A clustered Web server

The literature contains a considerable body of work on job scheduling (see [2, 8, 9, 10, 13] and references therein), but most of the work models the randomness of arrivals and job sizes using exponential-type distributions. When job sizes are *iid (independently identically distributed)* exponential and their arrivals follow a Poisson process, then the scheduling problem is fairly well understood [10, 11, 13, 17]. However, there is a great deal of evidence suggesting that the sizes of files traveling on the Internet do not follow an exponential-type distribution. Rather, these sizes appear to follow power-law distributions, which by definition satisfy the following equation:

$$\forall x \ \mathbb{P}[X > x] \quad \sim \quad \frac{c}{x^{\alpha}},$$

where X is a random variable, $c > 0$ and $1 \leq \alpha \leq 2$ (see [4, 1, 5]). For job sizes to follow a power-law distribution, intuitively means that a small fraction of jobs make up a large fraction of the overall load. Power-law distributions are heavy-tailed, and therefore, non-exponential. We mention that even when caching is used by Web servers, the file-size distribution appears to be heavy-tailed [4]. Web cluster scheduling, considering this type of distribution, has been addressed by a few researchers [7, 3, 12].

Our approach to the job scheduling problem differs in a fundamental way from traditional approaches, which *call for the dispatcher to assign jobs to servers as soon as they arrive.* In contrast, we propose in this paper that the dispatcher *hold back* and *not* assign requests to back-end servers on their arrival. Rather, we argue that is beneficial to allow the dispatcher to perform size-based re-shuffling of incoming requests. To this end, we propose a policy, dubbed `DAS` (Deferred Assignment Scheduling), where the dispatcher holds on to requests in its queue and makes a judicious assignment later on. More specifically, under this policy, the dispatcher assigns a request to a back-end server only when the server has finished the work assigned thus far. We show in this paper that if the dispatcher then selects the shortest job in its queue to be assigned to that server, then the average waiting time is reduced by as much as a factor of twenty, as compared with traditional policies.

In practice, the `DAS` policy is hard to realize for several reasons. First, it places a heavy burden on the dispatcher, which might have to maintain a potentially large number of requests in its queue. Second, it assumes that the dispatcher knows when back-end servers become idle, which requires extra communications overhead between the dispatcher and back-end servers. To mitigate these and other drawbacks of `DAS`, we introduce another policy, which we call `B-DAS` (Bounded Deferred Assignment Scheduling). Under `B-DAS`, the dispatcher assigns "short" jobs as soon as they arrive, while "long" jobs are assigned to back-end servers only when the servers become idle, or a certain time period has elapsed since the jobs were received. We argue that `B-DAS` corrects the drawbacks of `DAS` if job sizes follow power-law distributions. In this case, we can choose a cutoff point parameter that separates all jobs into short or long categories. A judicious choice of the cutoff point would ensure that only a few (long) jobs are not assigned immediately. Consequently, the burden on the dispatcher would be alleviated, and the overhead due to dispatcher-server communications would be greatly diminished. Experimental results show that if the dispatcher defers assignment for less than 4% of the jobs, then `B-DAS` still outperforms traditional policies to various degrees.

To gauge the performance of the proposed policies, we exercised them on empirical data traces measured at Internet sites serving the 1998 World Cup. We point out that Arlitt and Jin [1] have shown that job sizes from these traces follow a power-law distribution with $\alpha = 1.37$.

The rest of the paper is organized as follows: Section 2 presents related work in the literature. Section 3 discusses the `DAS` policy and provides a detailed performance analysis for it. Sections 4 discusses `B-DAS`, which is a practical implementation of `DAS`. Finally, Section 5 concludes the paper.

2 Previous Work

This section reviews briefly scheduling algorithms devised for exponentially distributed job sizes, as well as those that follow a power-law distribution.

Smith [15] considered scheduling with fixed-size (deterministic) jobs on a single server. The paper showed that in this case scheduling the shortest jobs first is optimal in that the algorithm gives minimal completion times. In a similar vein, Rothkopf [14] showed later that this algorithm is also optimal, in the same sense, for job sizes from a known distribution. Next, Winston [17] considered a clustered server with the first-come first-serve (FCFS) discipline at each server queue, exponential job-size distributions, and Poisson arrivals. The paper proved that under these assumptions the join-the-shortest-queue (JSQ) policy is optimal. However, Whitt [16] showed that there exist other job-size distributions for which JSQ is not optimal.

We next proceed with a review of scheduling policies for job sizes that follow a power-law distribution. Harchol-Balter et al. [7] devised a policy called "Size Interval Task Assignment with Equal Load" (SITA-E). SITA-E is based on the observation that when "short" jobs are stuck behind "long" jobs, then response time performance degrades. Such situations can be avoided if any back-end server is assigned only jobs of similar sizes. To this end, SITA-E 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

$$[s_0, s_1),\ [s_1, s_2),\ \cdots,\ [s_{n-1}, s_n)$$

are determined off-line so that each range contains approximately the same amount of work. Accordingly, when a request in the range $[s_j, s_{j+1})$ is received, the dispatcher assigns it to back-end server j. Under realistic job-size variance assumptions, [7] shows that SITA-E outperforms JSQ. In the same vein, Ciardo *et* *al.* [3] presents a load-balancing algorithm, called EquiLoad, and shows that it 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 algorithm, called AdaptLoad, is proposed in [12] as an adaptive, on-line version of EquiLoad. Still, AdaptLoad assigns each back-end server to a job-size range, but these ranges are continually re-evaluated based on the truncated history of requested jobs. The paper shows empirically that for very heavy load periods of World Cup traces, AdaptLoad outperforms JSQ—however, when the traffic is light or normal, JSQ outperforms AdaptLoad.

Table 1 displays a representative sample of scheduling policies.

Policy	**Dispatcher**	**Back-End server**
Uniform	Distributes jobs uniformly to back-end servers	FCFS (first come, first served)
Round-Robin (RR)	Assigns job i to back-end server $i \bmod n$	FCFS
Join Shortest Queue (JSQ)	Send jobs to servers with the least work load	FCFS, send dispatcher the queue size
SITA-E	Send jobs to servers for their size range. Equalize ranges	FCFS
EquiLoad	Distributes jobs by size to server for that size range	FCFS, dispatcher may readjust size ranges
AdaptLoad	Online sends jobs by size to server for that size range	FCFS

Table 1: Sample of clustered server scheduling policies

3 The DAS Policy

Web applications exhibit a mixture of task sizes spanning many orders of magnitude that reflect power-law distributions [7, 5, 1]. Consequently, a dispatcher may receive requests requiring service times of large variance. As mentioned before, it has been observed that when "short" jobs are stuck behind "long" jobs, then the overall waiting time increases, and, consequently, server performance degrades. To avoid this drawback, we propose the so-called `DAS` (Deferred Assignment Scheduling) policy, defined as follows:

> *The dispatcher does not distribute the requests (jobs) on their arrival. Rather, the dispatcher waits for a server to become idle, and then sends to it the shortest job that arrived up until that time.*

Recall that in contrast, traditional assignment policies assign requests to back-end servers upon arrival; then, each request is scheduled there for service according to some criteria. Intuitively, `DAS` would yield more efficient scheduling than traditional policies, because it utilizes superior information: The (global) information available to the dispatcher is superior to the (local) information available to an individual back-end server. More specifically, the dispatcher makes its scheduling decision based on *all* requests received thus far, while an individual back-end server essentially bases its decision only on the requests assigned to it.

We next proceed to illustrate the efficacy of `DAS` by comparing its performance with various other policies in two settings. First we present a brief motivational example, and then we show the results of a simulation driven by empirical data from the 1998 World-Cup.

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

Figure 2: A motivational example

3.1 A Motivational Example

We will compare `DAS` with the following policies:

1. `Round-Robin:` Jobs are assigned to back-end servers in a cyclical manner; namely, the i-th task is assigned to server $i \bmod n$, where n is the number of back-end servers in the cluster. 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 size range. This type of policy attempts to keep small tasks from getting stuck behind large jobs. Examples of this type of policy include SITA-E [7], EquiLoad [3], and to some degree, AdaptLoad [12].

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 job currently being served. By Winston [17], this policy is optimal when the job sizes follow an exponential-type distribution and have Poisson arrivals.

We now compute the average waiting time for the sequence of jobs presented in Figure 2, for a cluster with two back-end servers. 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. Let W_k^n denote the waiting time (excluding processing time) of job k at server n. Ignoring communications overhead, the waiting times at server S_1 are $W_1^1 = 0$, $W_3^1 = 1$, $W_5^1 = 99$, $W_7^1 = 99$, etc. Similarly, at server S_2, $W_2^2 = 0$, $W_4^2 = 1$, $W_6^2 = 1$, etc. Thus, the average waiting time for this web-cluster is:

$$\frac{\sum_{j=1}^{23} 99 + \sum_{j=1}^{24} 1}{50} \approx 46.$$

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}. It is worth noting, however, that server S_1 could not have scheduled any of the small jobs before job J_3. This is so, because at the time server S_1 commits to serve job J_3, that job is the only one available, with all other jobs yet to arrive at server S_1 after it had started processing job J_3.

Now consider the case that the dispatcher uses a Size-Range policy for assigning requests. Assume further that server S_1 is assigned jobs requiring service times in excess of 10 time units, while server S_2 receives the smaller jobs. This policy would give rise to the following assignment: server S_1 would be assigned job J_3, and server S_2 would be assigned all the other jobs. Notice that the load is evenly distributed among the two servers, each receiving jobs that require approximately 100 time units of service. In this scheme $W_1^2 = 0$, $W_2^2 = 2$, $W_3^2 = 3$, $W_4^2 = 4$, etc. The average waiting time is:

$$\frac{\sum_{j=2}^{50} j}{50} \approx 25.$$

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

A similar high waiting time is obtained if the dispatcher were to use the JSQ policy. In this case, server S_1 would be assigned jobs J_1 and J_3, while server S_2 would be assigned the rest of the jobs. The corresponding average waiting time is approximately 22.

Finally, consider a dispatcher that uses the `DAS` policy. Then 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 scheduled without delay, whereas the long job is scheduled last, yielding an average wait of approximately 1. This exceptionally low average waiting time is attained, because whenever a server becomes idle, the dispatcher has a small job on hand to assign to it. Consequently, small jobs are distributed equally between the two back-end servers, thereby affording them the opportunity to be served immediately.

3.2 Performance Study

We next demonstrate the superior performance of `DAS` by running a simulation driven by trace data from Internet sites serving the 1998 World-Cup. The data used was archived in an Internet repository (See [1] and http://ita.ee.lbl.gov/html/traces.html).

The Workload. The aforementioned repository provides detailed information about the 1.3 billion requests received by the sites over 92 days – from April 26, 1998 to July 26, 1998. The trace selected covers the first 600 minutes of June 26th and contains over 11 million requests. Figure 3 depicts the number of requests received by the server in minute intervals, and Figure 4 shows the number of bytes requested in each minute.

From each trace record, we extracted only the request arrival time and the size of the requested file. Since no information was recorded regarding the service time, our simulation experiments posited a service time proportional to the size of the requested document. We point out that this is an assumption whose reasonableness has been argued elsewhere [12].

It has been shown in Arlitt and Jin [1] that job sizes from World Cup traces follow a power-law distribution with $\alpha = 1.37$.

The Simulation. In order to evaluate the relative efficacy of various scheduling policies, we compare their performance with respect to the following statistics:

1. Average waiting time (excluding service time);
2. Average slow-down (the ratio of a job waiting time to its service time – file size; in our case)
3. The distribution of the number of requests that started service within a given time period after their arrival at the dispatcher. (More specifically, we compute the number of requests whose processing started within 5 ms from their arrival at the dispatcher, the number of requests whose processing started after 5 ms, but no later than 10 ms from their arrival at the dispatcher, etc.) This distribution captures the temporal dynamics of scheduling in terms of the delay from arrival to service commencement.

The scheduling policies compared are `DAS`, `Round-Robin` and `JSQ`. We did not, however, simulate the performance of Size-Range policies, because it has been shown in [12] that for this particular part of the trace used, they are slightly outperformed by the JSQ policy. The experiments considered a cluster with four back-end servers, and made the following assumptions: (1) communications times between the dispatcher and back-end servers, as well as the overhead incurred by the dispatcher to select a job/server, are negligible, (2) there is no job preemption.

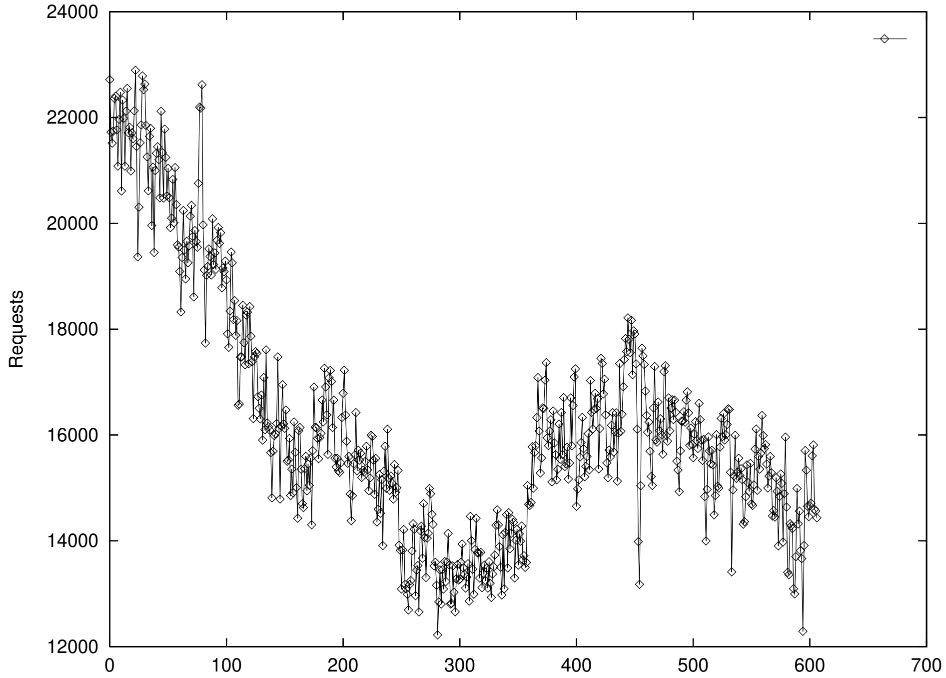

Figure 3: Number of requests arrivals per minute

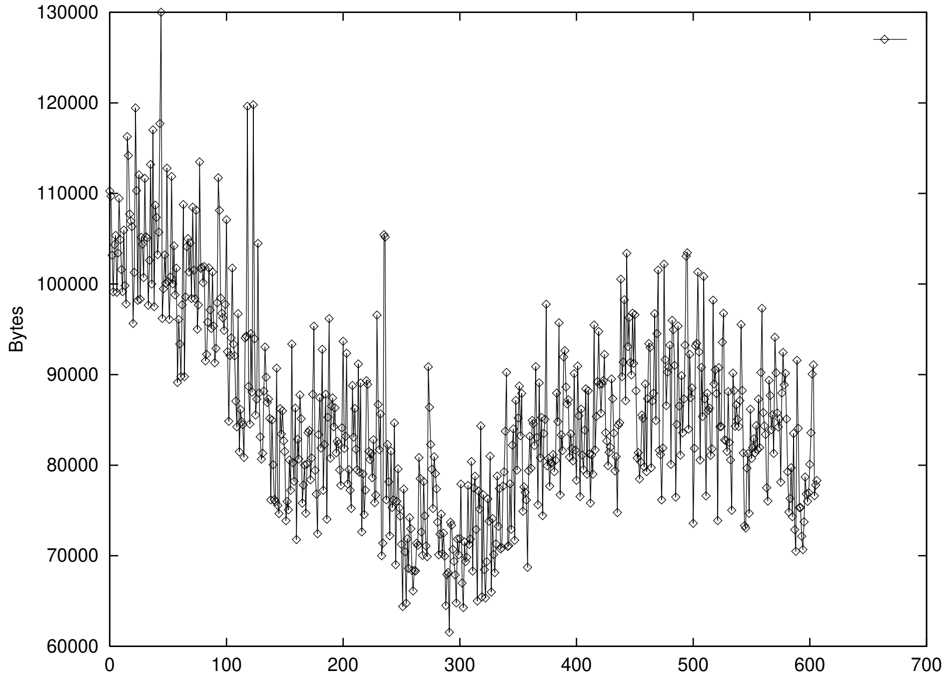

Figure 4: Total number of bytes requested per minute

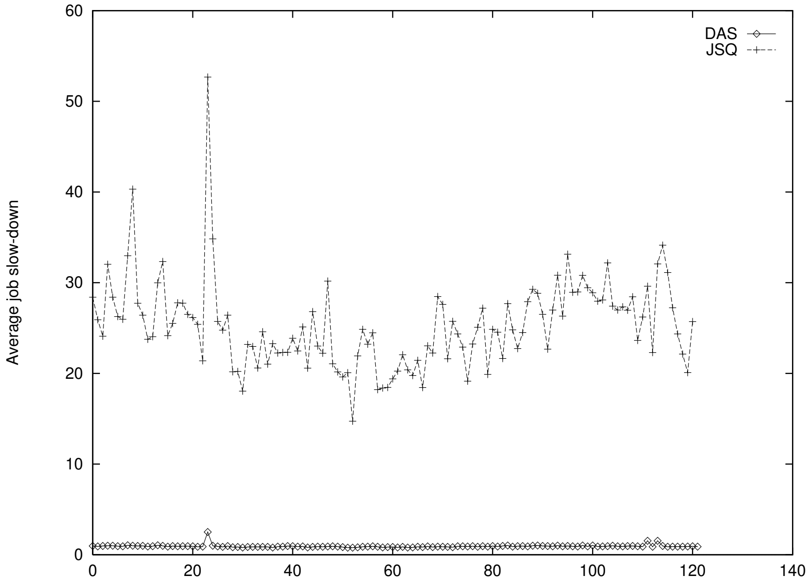

Figure 5: Average slow-down as function of time (time unit is 5 min.)

Figure 5 displays the average slow-down for policies JSQ and DAS, over successive 5-minute intervals. The results for the Round-Robin policy are not plotted, because it is outperformed by the JSQ policy. The figure shows that the DAS policy yields a substantial lower slow-down than the JSQ policy in all time intervals considered.

Finally, Table 2 displays the performance of the Round-Robin, JSQ and DAS policies for the overall simulation horizon. The results show again that the DAS policy performs substantially better than all other policies. More specifically, the average slow-down for DAS is 1, while the average slow-down time for the next best policy, JSQ, is 25. The Round-Robin strategy performs far worse, yielding an average slow-down of 74. The results confirm that the performance of a policy depends considerably on the amount of information acted on at the time of assignment and scheduling. The Round-Robin strategy (where the dispatcher has no knowledge of the expected service time) yields higher average waiting time and higher average slow-down than both JSQ and DAS. Indeed, for both JSQ and DAS, the dispatcher has complete information about the job sizes.

It is worth noting that in both DAS and JSQ, the dispatcher uses the same information. However, DAS outperforms JSQ by a factor of 25. We attribute this substantial improvement in performance to the fact that DAS does not require requests to be assigned immediately.

Policy	**Average Waiting Time (ms)**	**Average Slow-down**
Round-Robin	107	74
JSQ	36	25
DAS	5	1

Table 2: Comparative statistics for some clustered Web server scheduling policies

4 A Practical Implementation of DAS

While the `DAS` policy yields superior results, it unfortunately has some potentially serious implementation shortcomings. We now enumerate some of the difficulties attendant to a basic implementation of the `DAS` policy:

1. The dispatcher has to know precisely the service time of each job. `DAS` relies on the assumption that a job's service time is proportional to the file size associated with that job. This is indeed a reasonable estimate, but only when the file is not cached by the back-end server that processes the associated request. In fact, if the document is cached, then the service time may be reduced by as much as a factor of 10 [3]. Thus, in this case, the dispatcher may not, in fact, assign the shortest job first after all.

2. The dispatcher has to know when back-end servers are about to become idle. Again, because of caching, file sizes may yield a poor estimate of the actual service time. This implies, that a server has to explicitly notify the dispatcher when it becomes idle, and the attendant communications overhead is likely to hurt the performance of the server.

3. Long jobs may be delayed indefinitely.

In order to deal with these issues, we propose to refine the basic DAS policy into the so-called `B-DAS` (Bounded Deferred Assignment Scheduling) policy, by applying global scheduling only to long jobs. More specifically, policy `B-DAS` works as follows:

1. *The dispatcher classifies arriving jobs into long and short according to some prescribed cutoff point.*

2. *Short jobs are assigned in round-robin manner as soon as they arrive. Back-end servers process the jobs in their queue by scheduling the shortest job first, and notify the dispatcher when they finish the jobs assigned to them.*

3. *The dispatcher assigns long jobs in the following manner. First, if a back-end server becomes idle, it is assigned the shortest of the (long) jobs that have arrived up until that time. Second, a long job is assigned to some server, once it has been deferred more than a prescribed time interval threshold. In the latter case, the dispatcher assigns the long job to the back-end server with the least amount of work in its queue*[1]*. In either case, while the back-end server processes the long request, it will not be assigned any short jobs.*

We claim that if the service times follow a power-law distribution, then `B-DAS` remedies all the mentioned shortcomings of `DAS`. We justify this claim by the following arguments. First, in a power-law distribution, long jobs represent only a small fraction of all jobs, but a large fraction of workload. This in turn implies that a back-end server needs to notify the dispatcher that it is idle quite infrequently (recall that under `B-DAS`, a back-end server has no notify the dispatcher when it finishes a long job that was assigned to it). Second, the dispatcher's estimation of service time is far more accurate, because requests for large files occur infrequently compared with short ones. Thus, with high probability, the file is not cached by the back-end server. Finally, as its name suggests, the `B-DAS` policy imposes a *bounded* delay on long jobs waiting to be assigned to a server.

[1] The back-end server schedules this long job as soon as possible, that is, before any request that arrived after it (even if they are smaller).

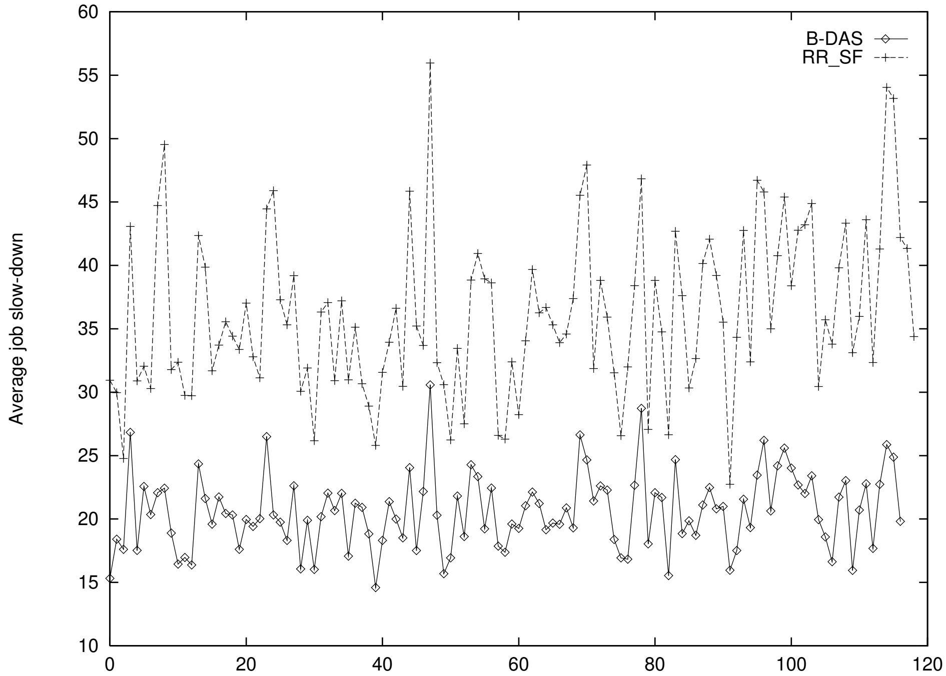

Figure 6: Average slow-down of B-DAS and RR_SF as function of time (time unit is 5 min.)

Experimental Results. The rationale underlying the B-DAS policy is the assumption that DAS performance will not change dramatically if only long jobs are deferred by the dispatcher. A case in point is our motivational example, where the B-DAS and DAS policies give rise to the same schedule, and consequently, yield the same performance. To test the validity of this assumption, we simulated the B-DAS policy, using the same World-Cup data. We chose, rather arbitrarily, the maximal delay of a long job at the dispatcher to be 1 second, and the cutoff between short and long jobs to be 20K. This cutoff value was chosen, because it leaves less than 4% of the jobs designated as "long".

In order to see the performance gain resulting from deferred assignment of long jobs, we compared the B-DAS policy with a similar one, dubbed RR_SF (Round-Robin Shortest First). RR_SF calls for all jobs to be assigned immediately on arrival to back-end servers in round-robin manner, and each server processes the shortest job in its queue first. Thus, the only difference between the B-DAS and RR_SF policies is their treatment of long jobs.

Figure 5 displays the average slow-down for policies RR_SF and B-DAS, over successive 5 ms intervals. The B-DAS policy achieves approximately a 40% reduction of slow-down in all time intervals considered, as compared to the RR_SF policy.

Finally, Table 3 displays the performance of the RR_SF and B-DAS over the entire simulation time horizon. The results show that the B-DAS policy outperforms the RR_SF policy. More specifically, both the average waiting time and slow-down resulting from B-DAS are approximately 40% lower than those resulting from RR_SF .

Discussion. We end this section with a comparison between B-DAS and JSQ. It can be seen from Tables 2 and 3 that B-DAS outperforms JSQ slightly on both performance metrics considered. This is a remarkable result given that the dispatcher requires far less information in B-DAS than in JSQ. More specifically,

Policy	Average Waiting Time (ms)	Average Slow-down
RR-SF	57	36
B-DAS	35	22

Table 3: Comparative statistics for the RR_SF and B-DAS policies

B-DAS requires the dispatcher to know the sizes of fewer than 4% of the files served by the server cluster, and the status of the back-end servers relatively *infrequently.* On the other hand, JSQ requires the dispatcher to know the precise sizes of *all* files, and the status of the back-end servers at *all* times. This appears to imply that the considerable difference in information needed by the dispatcher is balanced by a superior strategy, namely, the assignment deferral of very long jobs.

5 Conclusion

In this paper, we advocate a novel approach to the job scheduling problem. The dispatcher is no longer forced to assign requests to back-end servers upon request arrival; rather, the dispatcher may defer assignment by waiting to accumulate more information to great advantage. Indeed, we have shown experimentally that this approach results in excellent performance as compared to traditional approaches. More specifically, the proposed DAS policy performs far better than the JSQ policy, where both policies require the dispatcher to have complete, precise knowledge about job service time and status of back-end servers. Furthermore, in the proposed B-DAS policy, the dispatcher requires considerably less informations than the JSQ policy; nevertheless, B-DAS still outperforms JSQ slightly.

This approach, however, is probably not efficacious for every workload pattern. For example, if the workload is light, then an arriving request may encounter an idle server with high probability, and consequently, be processed without delay. In this case, simple policies, such as Round-Robin, would perform comparably to DAS (or better in view of the communications overhead incurred by the latter). The real challenge is to find a way to continually adapt the dispatcher policy to changing workload patterns. This challenge will be treated in future work.

References

[1] Martin Arlitt and Tai Jin. "Workload Characterization of the 1998 World Cup Web Site," *IEEE Network*, Vol. 14, No. 3, 30-37, May/June 2000. Extended version: Tech Report HPL-1999-35R1, Hewlett-Packard Laboratories, September 1999.

[2] Peter Brucker. *Scheduling Algorithms*, Third Edition, Springer-Verlag, 2001.

[3] Gianfranco Ciardo, Alma Riska, and Evgenia Smirni. "EquiLoad: a load balancing policy for clustered web servers", In *Performance Evaluation* 46(2-3): 101-124, 2001.

[4] Mark E. Crovella, Murad S. Taqqu and Azer Bestavros. "Heavy-tailed Probability Distributions in the World Wide Web," In *A Practical Guide To Heavy Tails*, Chapman Hall, New York, pp. 3–26, 1998.

[5] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. "On Power-Law Relationships of the Internet Topology," In Proceedings of *ACM SIGCOMM '99*, 251-262, Aug. 1999.

[6] Mor Harchol-Balter. “Task Assignment with Unknown Duration,” *Journal of the ACM*, Vol. 49, No. 2, 260-288, March 2002. (Extended Abstract in *20th International Conference on Distributed Computing Systems (ICDCS ’00), Taipei, Taiwan, April 2000.*)

[7] Mor Harchol-Balter, M. E. Crovella and C. D. Murta. “On Choosing a Talk Assignment Policy for a Distributed Server System,” In *Proceedings of Performance Tools ’98*, Lecture Notes in Computer Science, Vol. 1468, 231-242, 1998.

[8] Michael Katehakis and C. Melolidakis.“On Stochastic Optimality of Policies in First Passage problems”, In *Stochastic Analysis and Applications*, 8 (2) 12-25, 1990.

[9] Michael Katehakis and C. Melolidakis. “On The Optimal Maintenance of Systems and Control of Arrivals in Queues”, In *Stochastic Analysis and Applications*, 8 (2) 12-25, 1994.

[10] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, 2002.

[11] Rhonda Righter. “Scheduling in Multiclass Networks with Deterministic Service Times”. In *Queueing Systems* 41(4): 305-319, 2002.

[12] Alma Riska, Wei Sun, Evgenia Smirni, Gianfranco Ciardo . “AdaptLoad: Effective Balancing in Clustered Web Servers Under Transient Load Conditions,” in *22nd International Conference on Distributed Computing Systems (ICDCS’02)*, 2002.

[13] Sheldon M. Ross. *Probability Models for Computer Science*, Academic Press, 2002.

[14] Michael H. Rothkopf. “Scheduling with random service times” , In *Management Science*, 12;703-713, 1966.

[15] Wayne E. Smith. “Various optimizers for Single-Stage Production.” *Naval Research Logistics Quarterly*, Vol. 3, 59-66, 1956

[16] Ward Whitt. “Deciding which Queue to Join: Some Counter Examples,” *Operations Research*, Vol. 34, No. 1, 55-62, 1986.

[17] Wayne Winston. “Optimality of the Shortest Line Discipline,” *Journal of Applied Probability*, 14:181–189, 1977.