

# Performance-Guided Load (Un)Balancing Under Autocorrelated Flows

Qi Zhang<sup>1</sup>   Ningfang Mi<sup>1</sup>   Alma Riska<sup>2</sup>   Evgenia Smirni<sup>1</sup>

<sup>1</sup> Department of Computer Science, College of William and Mary, Williamsburg, VA

<sup>2</sup> Seagate Research, 1251 Waterfront Place, Pittsburgh, PA 15222

## Abstract

Size-based policies have been shown in the literature to effectively balance load and improve performance in cluster environments. Size-based policies assign jobs to servers based on the job size and their performance improvements are an outcome of separating “short” from “long” jobs, by avoiding having short jobs waiting behind long jobs for service. In this paper, we present evidence that performance improvements due to this separation quickly vanish if the arrival process to the cluster is autocorrelated. Based on our observations, we devise a new size-based policy called D\_EQAL that still strives to separate jobs to servers according to job size but this separation is now biased by an effort to reduce performance loss due to autocorrelation in the arrival flows of jobs that are directed to each server. As a result of this bias, all servers may not be equally utilized (i.e., load in the system may be “unbalanced”), but performance benefits become significant. D\_EQAL can be used on-line as it does not assume any a priori knowledge of the incoming workload. Extensive simulations show the effectiveness of D\_EQAL under autocorrelated and uncorrelated arrival streams and illustrate that the policy successfully self-adjusts the degree of load unbalancing based on monitored performance measures.

# 1 Introduction

In the past few years, there has been a renewed interest in the development of load balancing policies for clustered systems with a single system image, i.e., systems where a set of (homogeneous) hosts behaves as a single host. Jobs (or requests) arrive at a dispatcher which then forwards them to the appropriate server.<sup>1</sup> While there exists no central waiting queue at the dispatcher, each server has a separate waiting queue and a separate processor that operates under the first-come first-serve (FCFS) queueing discipline, see Figure 1. The dispatching policy is critical for system performance and strongly depends on the stochastic characteristics of the jobs as well as on the performance measures that the system strives to optimize. If job service times are highly variable, then policies that balance the load in the cluster using as a basic criterion the size of the incoming job, have been shown to minimize the expected job completion time and the expected job waiting time [10, 9]. This broad family of load balancing policies is known as *size-based* policies.

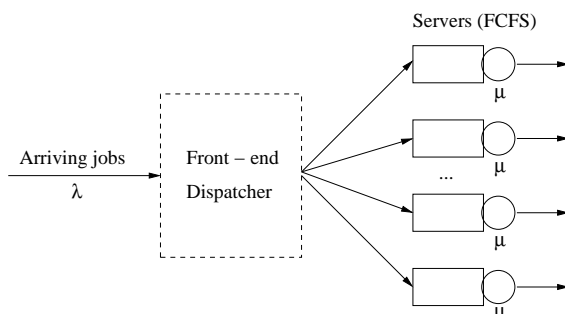


Figure 1: Model of a clustered server.

The basic premise for the success of size-based policies is reduction of variability in the job service time distribution at each server. First, it has been demonstrated in the literature that increased variability in the service process of an  $M/GI/1$  queue results in longer waiting queue lengths [13]. Longer waiting queues imply longer expected job response times (i.e., waiting plus service times) and longer average job slowdowns (defined as the expected value of the ratio of the job response time to the job service time). In an  $M/GI/1$  setting, the performance of small jobs that are queued behind large jobs degrades significantly, which then contributes to longer average job response time and longer average job slowdown. Size-based policies that direct jobs of similar sizes to the same

---

<sup>1</sup>In this paper, we use the terms “jobs” and “requests” interchangeably.

server aim at reducing the variation in job service times seen by each server and at diminishing the proportion of small jobs waiting behind long jobs. Defining the intervals of job sizes served by each server must be done judiciously. If there is a priori knowledge of the job service time distribution, then splitting the cumulative distribution function (CDF) of job service times in equal parts, as many as the servers, ensures that all servers are equally utilized (i.e., the system is load balanced) and that jobs of “similar” size are directed to the same server. The optimality of size-based policies in homogeneous environments with respect to minimizing the expected job response time and the expected job waiting time has been proved in [9].

Not all size-based policies require a priori knowledge of the job service time distribution as the empirical distribution may be estimated on-the-fly by collecting statistics of the past workload seen by the system [25]. A required condition though for size-based policies is that upon job arrival at the dispatcher, an accurate estimate of the job service time is possible. This condition restricts our discussion here to systems where accurate estimation of job service times is possible.

Several types of clustered systems can take advantage of size-based policies. One example is locally-distributed Web server clusters where a switch is the initial interface between the cluster nodes and the Internet [3, 25, 15]. For static requests in Web server clusters, e.g., transfers of image or text files, the job service time is analogous to the size of the transferred file. This information can be immediately used by the dispatcher to assign the request to the appropriate server [25].<sup>2</sup> Media-server clusters that provide streaming of audio and video is another example of a centralized cluster where job size is known a priori and where size-based policies can be used [22, 6]. Storage systems that use mirroring to improve performance and data availability is yet another case of a cluster system where load balancing based on the job size is possible.

Here, we focus on the general problem of load balancing in a homogeneous cluster of FCFS servers depicted in Figure 1, aiming at improving the expected job response time and the expected job slowdown. In contrast to prior work in size-based policies, which assumes that the arrival process at the dispatcher is independent and identically distributed (i.i.d.), we examine the performance of size-based policies under an *autocorrelated* arrival process. Autocorrelation in the arrival

---

<sup>2</sup>Size-based policies can be adapted for Web server clusters that serve dynamic requests. For details we direct the reader to [25]. Our focus here is on the more general problem of the effectiveness of size-based policies under autocorrelated arrivals.

process implies that there is a dependence structure in the arrival flows. Conventional wisdom has it that the arrival process in Internet servers is not i.i.d. because of the self-similar nature of network traffic [20]. Autocorrelated flows have also been observed in multi-tiered systems [14] and storage systems [17].

In this paper, we show that the effectiveness of size-based policies diminishes if the workload arrival process is autocorrelated. We closely examine the performance effects of autocorrelation under several load balancing policies including ADAPTLLOAD, a size-based policy, and several classic policies including *Join the Shortest Weighed Queue*, *Join the Shortest Queue*, and *Round-Robin* in order to build intuition on the problem. Based on our observations, we propose D\_EQAL, a new size-based load balancing policy that reduces performance degradation due to autocorrelation in each server. D\_EQAL dynamically distributes work equally addressing autocorrelation and load, and *may* unbalance load in the system in order to benefit performance. If the arrival process to the cluster is uncorrelated, then the policy loads each server with equal work, i.e., aims at balancing the load across all servers. If the arrival process to the cluster is autocorrelated, then D\_EQAL loads each server with unequal work such that load in the system becomes unbalanced, but overall system performance increases dramatically. D\_EQAL does not assume any a priori knowledge of the job service time distribution nor any knowledge of the autocorrelation structure in the arrival streams, yet it successfully copes with changing workloads by observing past arrival and service characteristics as well as past performance. To the best of our knowledge, this is the first time that dependence in the arrival process becomes a critical aspect of load balancing.

This paper is organized as follows. Section 2 presents related work. Section 3 gives evidence of performance deterioration in a single server system due to autocorrelated arrivals and demonstrates that performance gains of size-based policies in clusters quickly evaporate in the presence of autocorrelated arrivals. In Section 4 we first present a policy that unbalances the load in a static manner to improve performance under autocorrelated flows. D\_EQAL, the proposed on-line size-based policy, is presented later in the section and its performance is evaluated via detailed simulation. Section 5 summarizes our contributions.

## 2 Related Work

A significant body of research in task scheduling and load balancing has been developed over the years (see [11, 9] and references within), but only recently there has been a consensus that traditional load balancing policies, i.e., join-the-shortest queue or join-the-least-loaded server, result in high average job response time and high average job slowdown if job service times are highly variable and/or heavy-tailed [10]. For workloads with highly variable service times, size-based policies, which advocate dedicating servers to jobs of similar sizes, have been shown in the literature to achieve high performance [10, 25]. Assuming that there are  $N$  servers, the job sizes are partitioned into  $N$  intervals,  $[s_0 \equiv 0, s_1)$ ,  $[s_1, s_2)$ ,  $\dots$ ,  $[s_{N-1}, s_N \equiv \infty)$ , so that server  $i$  is responsible for satisfying requests of size between  $s_{i-1}$  and  $s_i$ . By dedicating servers to requests of similar size, these policies reduce the average job slowdown through separation of long and short jobs. Despite the fact that size-based policies are stateless, i.e., oblivious of the instantaneous load in each server, they successfully load each server with approximately the same amount of work so that all servers are equally utilized [10, 9]. The optimality of size-based strategies is proved in [9].

Note that size-based policies are based solely on a priori knowledge of the distribution of the incoming job sizes and not of the instantaneous load in the servers. Even if the job service time distributions are *not* known a priori, on-line versions of size-based policies have shown to maintain high performance for workloads that are highly variable *across time*, i.e., transient workloads [25]. ADAPTLLOAD has been developed as an on-line version of a size-based policy that monitors the incoming workload and builds a histogram of job size frequency (i.e., builds the empirical distribution histogram) while the system is in operation. Based on this histogram, it self-adjusts the interval boundaries according to changes in the operational environment. Rapid fluctuations in job size frequencies and/or job service times are now reflected in the histogram.

Size-based load balancing policies have been examined under the assumption of i.i.d. arrivals into the cluster. Nonetheless, there is a significant body of literature that shows that dependence in arrival flows exists, especially in network-related traffic [20]. Autocorrelated flows have been also observed in multi-tiered systems [14] and storage systems [17]. Even for systems that operate under low to moderate utilization levels, increased autocorrelation in their arrival process has been shown detrimental for performance, i.e., the higher the autocorrelation, the longer the expected

response times [8]. Similar results are reported in [2] where the performance effects of short-range dependence versus long-range dependence in the arrival streams are examined. In the context of networking, traffic shaping has been used as a technique to alleviate the negative effects of autocorrelation, by dropping, reordering, or delaying selected requests [5, 21, 7, 1]. Finally, recent analytic models of a single queue with autocorrelated inter-arrival and/or service process, have demonstrated that flows out of the queue are also autocorrelated and propagate to the next queue that feeds from that departure process [23].

### 3 Motivation

In this section, we first present data that have been measured on real systems to confirm the existence of dependence in arrival streams. Then, we give motivation for this work first by presenting the performance of a single server under autocorrelated arrivals and second by examining the performance of load balancing policies in a cluster under autocorrelated arrivals.

#### 3.1 Autocorrelation in Systems

Throughout this paper, we use the autocorrelation function (ACF) as a metric of the dependence structure of a time series (either request arrivals or services) and the coefficient of variation (CV) as a metric of variability in a time series (either request arrivals or services). CV values less than 1 indicate that the variability of the sample is low. CV values larger than 1 indicate high variability. The exponential distribution has a CV of 1.

Consider a stationary time series of random variables  $\{X_n\}$ , where  $n = 0, \dots, \infty$ , in discrete time. The autocorrelation function (ACF)  $\rho_X(k)$  and the coefficient of variation (CV) are defined as follows

$$\rho_X(k) = \rho_{X_t, X_{t+k}} = \frac{E[(X_t - \mu)(X_{t+k} - \mu)]}{\sigma^2}, \quad CV = \frac{\sigma}{\mu},$$

where  $\mu$  is the mean and  $\sigma^2$  is the common variance of  $\{X_n\}$ . The argument  $k$  is called the lag and denotes the time separation between  $X_t$  and  $X_{t+k}$ . The values of  $\rho_X(k)$  may range from -1 to 1. If  $\rho_X(k) = 0$ , then there is no autocorrelation at lag  $k$ . If  $\rho_X(k) = 0$  for all  $k > 0$  then the time series is uncorrelated. In most cases, ACF approaches zero as  $k$  increases. The ACF essentially

captures the “ordering” of random values in the time series. Positive ACF values imply that there is strong temporal locality, i.e., a value of the random variable has a high probability to be followed by another variable of the same order of magnitude, while negative ACF implies the opposite. The ACF’s decay rate determines if a process exhibits weak or strong correlation.

Figure 2 presents the ACF of arrivals at several storage systems supporting (dedicatedly) various applications [16, 17]. The figure shows that the dependence structure in the request arrival streams to the storage systems differs among the systems that support different applications

This dependence structure is a result of multiple factors including the architecture of the storage system, the file system, and the resource management policies at all levels of the I/O path. For more evidence of the existence of autocorrelated flows in storage systems, see [17].

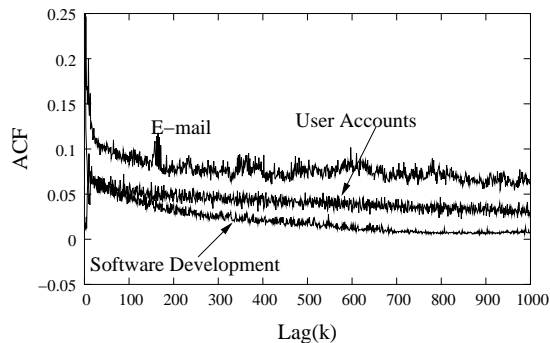


Figure 2: ACF of the arrival process at the storage subsystem for various applications.

### 3.2 Autocorrelation Effects in a Single Queue

To illustrate the magnitude of the performance effects of autocorrelation in systems, we parameterize a simple queueing model of a single server. The arrival process is drawn from a Markov Modulated Poisson Process (MMPP) [12] that is parameterized such that it results in three levels of dependence as illustrated in Figure 3(a): NOACF (i.e., arrivals are uncorrelated),  $ACF_1$ , and  $ACF_2$ . The probability distribution functions (PDFs) of these three arrival processes are identical (i.e., all their moments are the same), but what distinguishes them is the *order of sampling* from the PDF, which introduces autocorrelation. The mean inter-arrival time in these three processes is equal to 13.28 ms and CV is equal to 5.67, as derived by the arrival process to the storage system of a Web server presented in [17].

The service process is drawn from a 2-stage hyperexponential ( $H_2$ ) distribution with mean service time equal to 3 ms and CV equal to 1.85 and models the disk-level service process for the Web server storage trace in [17]. Inter-arrival times are scaled so that we examine system performance under different utilization levels.

Figures 3(b)-(d) present performance measures for the three different arrival processes as a function of server utilization. The effect of ACF on system performance is tremendous: the higher the ACF, the worse the system performance, which can worsen by as much as 3 orders of magnitude when comparing to the case with uncorrelated (NOACF) arrivals. Because of the difference in the three curves, the performance measures with uncorrelated arrivals look flat. Under uncorrelated arrivals (i.e., the NOACF curve), queue length, as expected, is equal to 152 for utilization equal to 0.9. This number is dwarfed in comparison to the corresponding values for the  $ACF_1$  and  $ACF_2$  curves. The inset plots in Figures 3(b)-(d) illustrate the same performance measures but using logarithmic scale on the  $y$  axis. The dramatic effects of autocorrelation are illustrated even for low to moderate system utilizations, between 25% and 50%, where the probability that a job finds the system idle is higher than 0.5.

It is the burstiness in the arrival stream that results in performance degradation by several orders of magnitude, even for low to moderate loads. This burstiness is captured by the autocorrelation metric. Positive ACF values greater than zero for lag  $k \geq 1$  imply that a small inter-arrival time has high conditional probability to be followed by a small  $k$ -th inter-arrival, causing the queue to build up fast. The stronger the dependence, the more the burstiness in the arrival stream, which causes the waiting queue to build up faster if the arrival stream is  $ACF_2$  versus  $ACF_1$ . If the arrival process is uncorrelated, the conditional probabilities are zero, i.e., there is no burstiness in the process, which implies less waiting queue build up (and consequently better performance) even for the same server utilization level as Figure 3 illustrates. Although bursty periods are relatively short, their impact on performance is long-term, as Figure 3 indicates. Moreover, if the goal in the system is to achieve a certain performance, then the system utilization should be kept at different levels for arrival processes with the same average behavior (i.e., mean and CV) but different dependence structures.



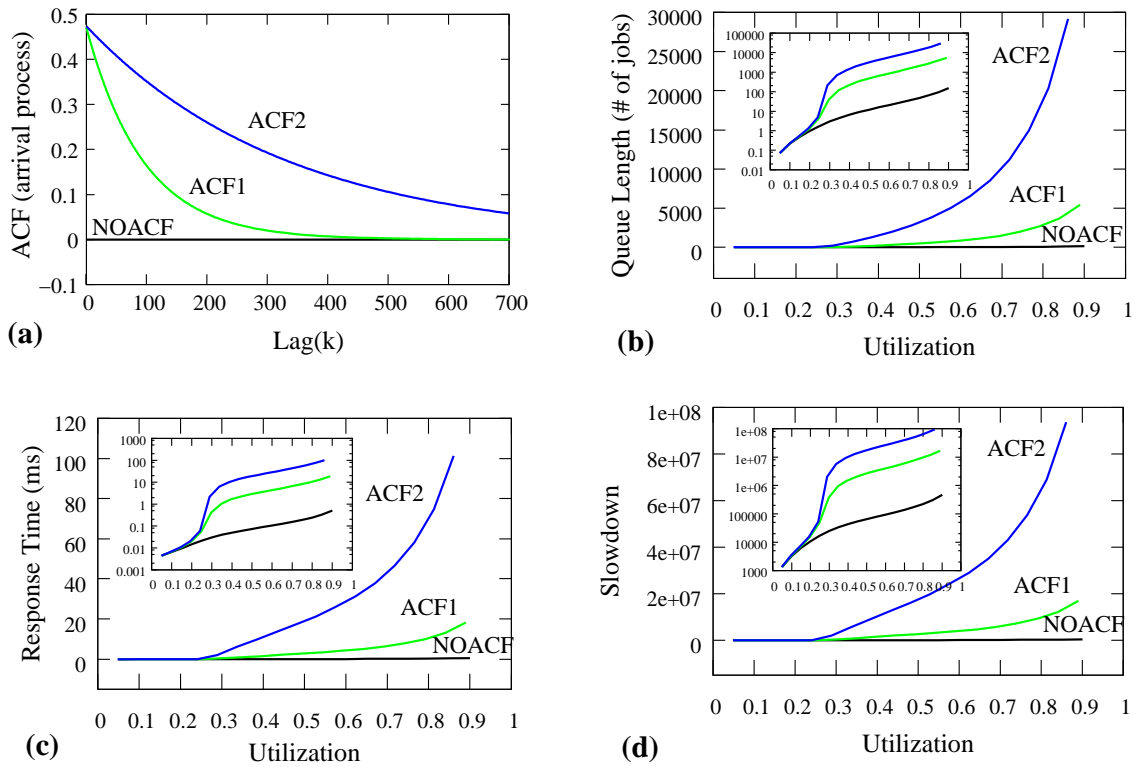


Figure 3: (a) ACF of the arrival process at the server, (b) average queue length, (c) average response time, and (d) average slowdown.

### 3.3 Autocorrelation Effects on Load Balancing Policies

In this section, we use simulation to examine the performance impacts of autocorrelated arrivals in load balancing policies in the cluster of Figure 1. We assume that the number of nodes is equal to four. Experiments with larger number of nodes have been also done and results are qualitatively the same as those reported here.

While the traces in Figure 2 indicate that in a clustered system arrivals have different degrees of correlation, we do not have a detailed description of the underlying system [16]. This prohibits us from using those traces to drive our simulation. We opted to use another publicly available trace measured in a Web server cluster. Specifically, the service process is obtained from traces of the 1998 World Soccer Cup Web site,<sup>3</sup> that have been used in several load balancing studies [25, 18, 19]. Trace data were collected during 92 days, from April 26th 1998 to July 26th 1998 [4]. Here,

<sup>3</sup>Available from the Internet Traffic Archive at <http://ita.ee.lbl.gov>.

we use part of the June 24th trace (10 million requests), that corresponds to nearly ten hours of operation and we extract the file size of each transferred request. Because the Web site contained only static pages, the size of the requested file is a good approximation of the request service time. The average size of a requested file is 5059 bytes and its CV is 7.56. High variability in the file size distribution and file popularities that change dramatically over time, make this trace particularly challenging for load balancing and an excellent candidate to evaluate the performance of size-based policies, for more discussion on this trace see [25].

Unfortunately, we cannot use the arrival process of the World Cup trace data because it is not detailed enough: arrival timestamps of requests are provided in seconds, as a result there are *multiple* requests that arrive within one second periods. To examine the effect of autocorrelation in the arrival process, we use the three arrival processes generated by the MMPP process described in the previous subsection. Their autocorrelation structure is depicted in Figure 3(a).

We compare the performance of the following policies: ADAPTLLOAD, a size-based policy that uses a histogram of job sizes which is built on-line and has been shown to be effective under changing workload conditions [25], *Join the Shortest Weighed Queue* (JSWQ) [25], *Join the Shortest Queue* (JSQ) [13], and *Round-Robin* (RR).

- **ADAPTLLOAD:** In a cluster with  $N$  server nodes, ADAPTLLOAD partitions the possible request sizes into  $N$  intervals,  $\{[s_0 \equiv 0, s_1), [s_1, s_2), \dots [s_{N-1}, s_N \equiv \infty)\}$ , so that if the size of a request falls in the  $i$ -th interval, i.e.,  $[s_{i-1}, s_i)$ , this request is routed to server  $i$ , for  $1 \leq i \leq N$ . These boundaries  $s_i$  for  $1 \leq i \leq N$  are determined by constructing the histogram of request sizes and partitioning it in equal areas, i.e., representing equal work for each server, as shown by the following equation:

$$\int_{s_{i-1}}^{s_i} x \cdot dF(x) \approx \frac{\bar{S}}{N}, \quad 1 \leq i \leq N, \quad (1)$$

where  $F(x)$  is the cumulative distribution function (CDF) of the request sizes and the amount of total work is  $\bar{S}$ . By sending requests of similar size to the same server, the policy improves average job response time and average job slowdown by avoiding having short jobs being stuck after long jobs in the queue. For a transient workload, the values of the  $N - 1$  size boundaries  $s_1, s_2, \dots, s_{N-1}$  are critical. ADAPTLLOAD self-adjusts these boundaries by

predicting the incoming workload based on the histogram of the last  $K$  requests. In our simulations, we set the value of  $K$  equal to 10,000. For a detailed discussion on the policy sensitivity to this parameter, we direct the interested reader to [25].

- **JSWQ:** The length of each queue in the system is weighed by the size of each queued request, therefore each incoming request is routed to the least loaded server.
- **JSQ:** When a request arrives, it is assigned to a server with the smallest waiting queue. If multiple servers have the same queue length, then a server is selected randomly from this group of servers.
- **RR:** In the round-robin algorithm, requests are routed to servers in a rotated order.

ADAPTLLOAD is a size-based policy that has been shown in the literature to balance load effectively in workloads with highly variable service times [25]. Here, we show that if arrivals are autocorrelated, then ADAPTLLOAD’s performance is comparable to that of classic policies that are well-known to perform poorly.

For all simulation experiments presented in this paper, the arrival process in the cluster is synthetically generated using the three MMPP processes used in the previous subsection. The service process directly uses the World Cup trace data of June 24. In all our experiments, we consider a cluster of four homogeneous back-end servers that serve requests in a FCFS order. We report on the average job response time, average slowdown, average queue length, and average system utilization. Figure 4 plots performance results for the four load balancing policies. Similar to the results of Section 3.1, Figure 4 shows that autocorrelation in the arrival process degrades overall system performance for all four policies. Observe that overall performance under uncorrelated arrivals (NOACF) is two orders of magnitude better than under  $ACF_1$  inter-arrivals, and three orders of magnitude better than under  $ACF_2$  inter-arrivals, despite the fact that average overall system utilizations are exactly the same for all experiments, i.e., average utilizations are about 62%, see Figure 4(d). This is consistent with results presented in Section 3.1 for the single queue case. Per server utilizations, for all experiments, remain the same and equal to about 62%, but are not reported here due to lack of space. More importantly, the figure also shows that ADAPTLLOAD outperforms all policies under uncorrelated arrivals *only*, see Figures 4(a)-(c). Under correlated arrival

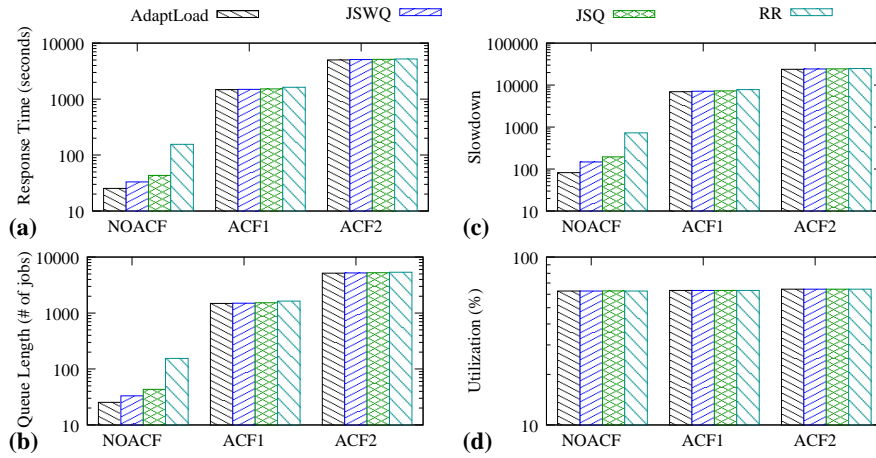


Figure 4: Performance metrics under four load balancing policies: (a) average response time, (b) average queue length, (c) average slowdown, and (d) average utilization.

processes, ADAPTLLOAD’s performance is comparable to that of the three other policies, because, in such conditions, excessive waiting in queue rather than load balancing decisions determine performance.

To better understand this behavior, we examine the autocorrelation of the arrival process in *each* server. Figure 5 shows the ACF of the arrival process at each back-end server, as well as the ACF of the arrival process at the front-end dispatcher (labeled as “original stream” in the figure). When there is no autocorrelation in the inter-arrivals at the front-end dispatcher (left column of graphs in Figure 5), the ACF of inter-arrivals at each back-end server is almost zero for all policies, except ADAPTLLOAD as captured in Figure 5(a).

The middle column of graphs in Figure 5 shows the ACFs for the experiment with the  $ACF_1$  structure in the arrival process, and the right column of graphs in Figure 5 shows the ACFs for the experiment with the  $ACF_1$  structure in the arrival process. JSWQ and JSQ have the weakest dependence while RR has the strongest dependence across all servers. The main difference between ADAPTLLOAD and the other three load balancing policies is that, under ADAPTLLOAD, the dependence structure in the arrival streams of different servers is different.

Because ADAPTLLOAD is a size-based policy and the workload is heavy-tailed, most requests are small and are directed to the first two servers. Specifically, the first server receives 88.6% of jobs, and the second server receives 8.7% of jobs. The remaining (large) jobs are sent to the third

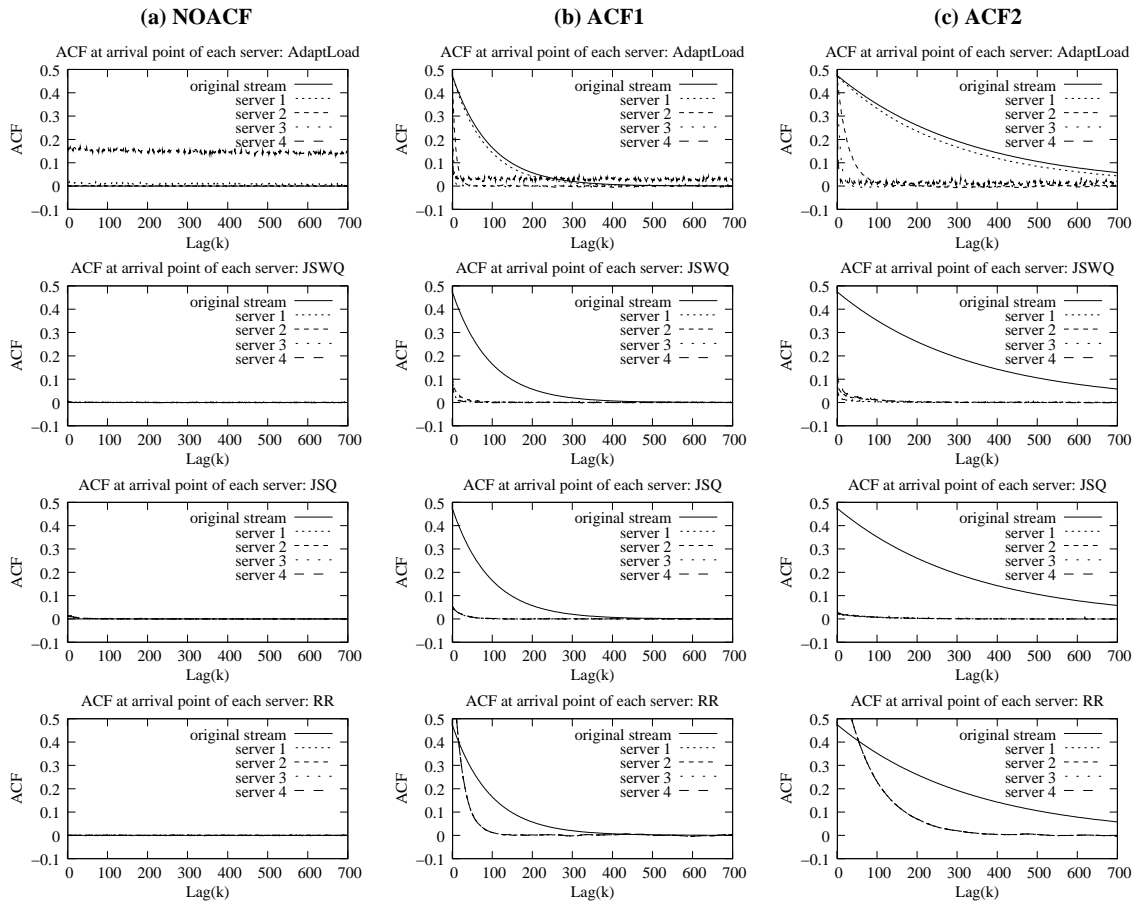


Figure 5: ACF in inter-arrivals at each server, where the arriving requests at the front-end dispatcher have (a) no dependence (NOACF), (b)  $ACF_1$  dependence, and (c)  $ACF_2$  dependence.

and fourth servers. Consequently, the first server inherits the dependence structure of the entire arrival stream. The arrivals at the second server are also autocorrelated, but at a lesser degree than at the first server. ACF in the arrivals of the first two servers does not affect their utilization, which remains almost the same as the rest of the servers. The rest of performance measures, however, are different for different servers, under ADAPTLLOAD. Looking at the per-server performance measures, we observe that, under ADAPTLLOAD, the performance of the first server (and proportionally of the second server) is much worse than the performance of the rest of the servers, negatively affecting overall performance measures for the entire system. This behavior is consistent with the single queue performance, see Figure 3, where performance measures for utilization around 60% differ by several orders of magnitude between arrival processes with different dependence struc-

tures. Weak ACFs in the arrival processes of all servers under JSWQ/JSQ help performance, but because short and long jobs are now served on the same server, performance measures remain low. The performance of RR suffers from both autocorrelated arrivals and mixing of long with short jobs on the same server, resulting in the worst performance of the examined policies.

The above observations suggest that in clusters with autocorrelated arrivals, unbalancing the load while maintaining the property of serving jobs of equal size in the same server (as under ADAPTLLOAD), may improve performance. Specifically, the only distinction among the servers in the cluster, under ADAPTLLOAD which balances load and work, is the correlation structure of the arrival streams, which implies that to achieve equal performance among these servers, their load should be unequal, as Figure 3 indicates. Because under the other three load balancing policies, the correlation structure of the arrival streams to the servers of the cluster is the same, load unbalancing will not help there to improve performance. In the next sections, we present two policies that are built on top of ADAPTLLOAD, one static and one dynamic, and aim at reducing the load of the server(s) that admit arrival streams with high autocorrelation in an effort to improve the performance of individual servers and consequently overall system performance.

## 4 Unbalancing Load to Improve Performance

First, we present S\_EQAL, a variation of ADAPTLLOAD, where the load of the servers with autocorrelated arrivals is reduced by a static percentage. Then, we present D\_EQAL, a dynamic version of the same policy, where the degree of load unbalancing is automatically re-adjusted to account for fluctuations in the incoming workload characteristics and improve policy performance seamlessly.

### 4.1 S\_EQAL: Static Policy

Recall that in an  $N$ -server cluster, ADAPTLLOAD assigns to each server  $\bar{S}/N$  of the work, provided that the amount of total work is  $\bar{S}$ . ADAPTLLOAD determines the boundaries  $s_i$ , of job sizes for each server  $1 \leq i \leq N$  by constructing the histogram of job sizes and partitioning it in  $N$  equal areas, see Eq. (1). This histogram is built efficiently on-the-fly with only a small space cost for its storage [25].

Analysis of per-server performance measures show that equally partitioning the histogram guarantees equal utilization of all servers. However, this may hurt performance of jobs that are directed to servers with correlated arrivals. With ADAPTLLOAD, the first server admits 88.6% of the jobs, thus it inherits the correlation structure of the entire arrival stream to the dispatcher. According to the single server analysis, this server should operate under a lower utilization level than the rest of the servers. Work must be shifted away from it, in order to reach similar performance levels with other servers, whose arrival stream is less correlated. Naturally, the work that is shifted away from the first server must be redistributed appropriately to the rest of the cluster. This observation is the basis of S\_EQAL, a new policy that still builds the histogram of job sizes as ADAPTLLOAD does, but sets new boundaries,  $s'_i$ , by weighting the work assigned to each server as a function of performance degradation due to autocorrelation of the (new) arrival process to each server. That is, servers that admit autocorrelated arrival streams must now be less loaded than those that admit streams that are uncorrelated.

Aiming at unbalancing the load across servers, we introduce a shifting percentage vector  $\mathbf{p} = (p_1, p_2, \dots, p_N)$ , so that the work assigned at server  $i$  is now equal to  $(1+p_i)\frac{\bar{S}}{N}$ , for  $1 \leq i \leq N$ . The elements of vector  $\mathbf{p}$  can take both negative and positive values. A negative  $p_i$  makes the amount of work assigned at server  $i$  less than its equal share of  $\bar{S}/N$ . A positive  $p_i$  makes the amount of work assigned at server  $i$  higher than its equal share of  $\bar{S}/N$ . Because  $\mathbf{p}$  simply shifts work from one server to another, it should satisfy  $\sum_{i=1}^N p_i = 0$ , for  $1 \leq i \leq N$ . The following equation formalizes this new load distribution:

$$\int_{s_{i-1}}^{s_i} x \cdot dF(x) \approx (1+p_i)\frac{\bar{S}}{N}, \quad 1 \leq i \leq N. \quad (2)$$

Figure 6 gives an illustration of the high level idea of this new policy.

S\_EQAL statically defines  $p_i$  for  $1 \leq i \leq N$ , by letting  $p_1$  be equal to a (negative) pre-determined initial shifting value  $R$ , i.e.,  $p_1 = -R$ . The rest of the shifting percentages  $p_i$ , for  $2 \leq i \leq N$ , are calculated using the algorithm of Figure 7.<sup>4</sup> Here, because the majority of the requests is small and they are directed to the first server, the smaller the  $i$ , the less work should be

---

<sup>4</sup>We stress that different algorithms can be used to determine how this shifting of load is done, provided that  $\sum_{i=1}^N p_i = 0$  for  $1 \leq i \leq N$ . Finding the *optimal* algorithm to set the shifting vector  $\mathbf{p}$  is out of the scope of this paper.

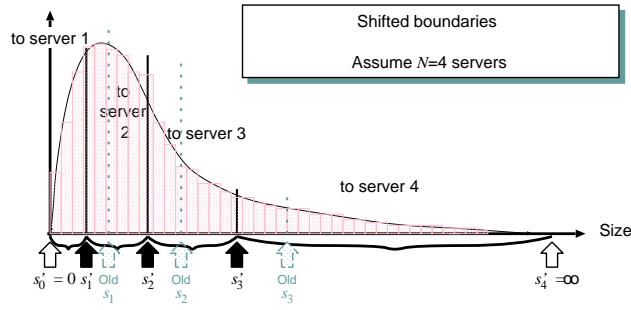


Figure 6: S\_EQAL's high level idea of boundary shifting.

dispatched to server  $i$ . This implies that  $p_1$  is negative. For server  $i$ , for  $1 \leq i < N$ , the portion  $R/2^{i-1}$  of its assigned work is equally distributed among servers  $i + 1$  to  $N$ . Initially, all shifting percentages  $p_i$  for  $1 \leq i \leq N$  are initialized to 0 (i.e., no shifting). For server 1,  $p_1$  is reduced by  $R$  (see 2.a in Figure 7). The work that is shifted from server 1 is now equally distributed among the remaining servers, i.e.,  $p_2, \dots, p_N$  increase by  $\frac{R}{N-1}$  (see 2.b in Figure 7) such that the condition  $\sum_{i=1}^N p_i = 0$  is satisfied. For server 2, the work shifted away from this server is now equal to  $\frac{R}{2}$  (see 2.c in Figure 7), so  $p_2$  is equal to  $\frac{R}{N-1} - \frac{R}{2}$ . The algorithm continues to equally distribute all shifted work from server 2 to the remaining servers. The iteration continues for servers 3, 4,  $\dots, N - 1$ . For example, if we define  $R = 10\%$ , i.e.,  $p_1 = -0.1$ , then according to S\_EQAL shifting percentage vector  $\mathbf{p} = (-10\%, -1.67\%, 3.33\%, 8.34\%)$ .

- |   |   |
|---|---|
| <b>1. initialize variables</b>                            |   |
| <b>a. initialize a variable <i>modify</i></b>             | $modify = R$                              |
| <b>b. initialize the shifting percentages</b>             | $p_i = 0$ for all $1 \leq i \leq N$       |
| <b>2. for <math>i = 1</math> to <math>N - 1</math> do</b> |   |
| <b>a. reduce <math>p_i</math> by <i>modify</i></b>        | $p_i \leftarrow p_i - modify$             |
| <b>b. for <math>j = i + 1</math> to <math>N</math> do</b> |   |
| equally distribute <i>modify</i> to the remaining servers | $p_j \leftarrow p_j + \frac{modify}{N-i}$ |
| <b>c. reduce <i>modify</i> to half</b>                    | $modify \leftarrow modify/2$              |

Figure 7: Setting the shifting percentages  $p_i$  for S\_EQAL.



### 4.1.1 Weakly correlated arrival process, ( $ACF_1$ )

First, we evaluate the performance of S\_EQAL with the  $ACF_1$  arrival process used in the previous section. We quantify the performance effect of different shifting vectors  $\mathbf{p}$  by presenting the aver-

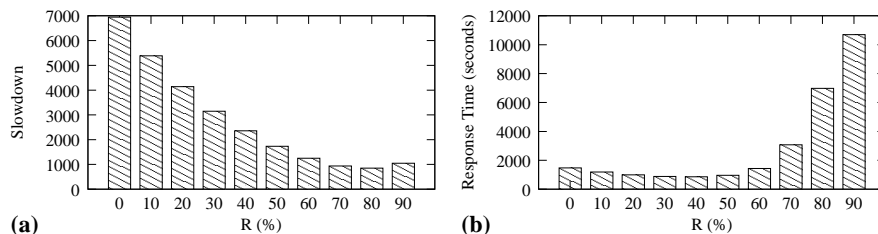


Figure 8: Average slowdown and average response time as a function of the initial shifting value  $R$  under  $ACF_1$  inter-arrival times.

age slowdown and average response time of requests under S\_EQAL for different initial shifting values  $R$ . Results are presented in Figure 8.  $R = 0\%$  corresponds to the original ADAPTL0AD, i.e., no boundary shifting. Figure 8(a) shows that the average slowdown of all requests improves as  $R$  increases (i.e., boundaries are shifted to the left). Best average slowdown is achieved for  $R = 80\%$  (i.e.,  $p_1 = -80\%$ ). Figure 8(b) shows that average response time increases for  $R > 40\%$ . Therefore, a good initial shifting value is  $R = 40\%$ , where average slowdown improves by 75.1% and average response time improves by 41.9% comparing to ADAPTL0AD, i.e.,  $R = 0\%$ .

We present the per-server performance in Figure 9. Per server utilizations shown in Figure 9(d) verify that the shifting percentages  $p_i$  indeed imbalance work across the cluster. As  $R$  increases, the utilizations of the first two servers decrease while utilizations of the last two servers increase. The last server's utilization is now the highest in the cluster. Reducing utilization in the first server reduces its request slowdown, as shown in Figure 9(a), but the extra work that is now assigned to servers 3 and 4 does not increase their slowdown significantly for small values of  $R$ . For  $R = 90\%$ , job slowdown at server 4 becomes very high, almost twice as high as for server 1, under ADAPTL0AD. Average per-server queue length behaves similarly to average slowdown, see Figure 9(c). The average response time displayed in Figure 9(b) shows that small  $R$  values decrease average response time of the first server and increase the response time of the last server. If the portion of additional requests served by the last server is small, then the contribution of the last server per-

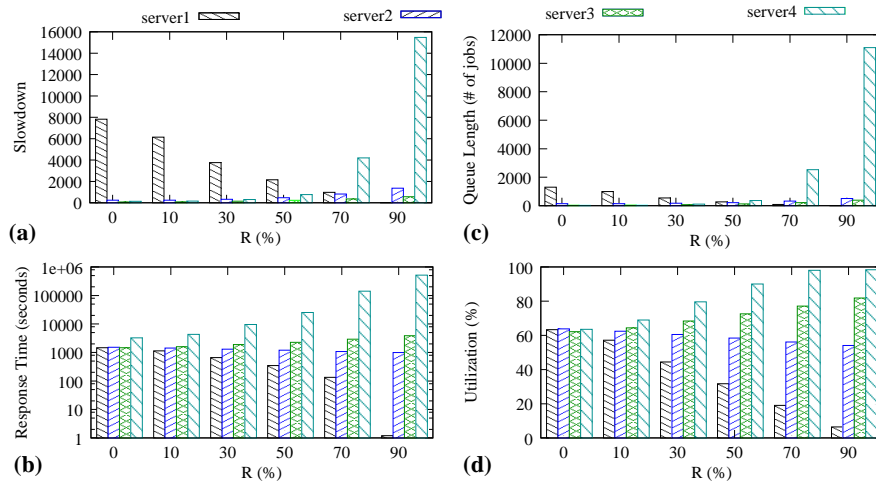


Figure 9: Per server performance measures: (a) average slowdown, (b) average response time, (c) average queue length and (d) average utilization as a function of the initial shifting value  $R$  with  $ACF_1$  inter-arrival times. The order of bars for each policy reflects server identity.

formance to the overall performance degradation is not significant. As  $R$  increases, more jobs are assigned to servers with larger index, which counterbalances the benefits of reducing utilization at the first two servers.

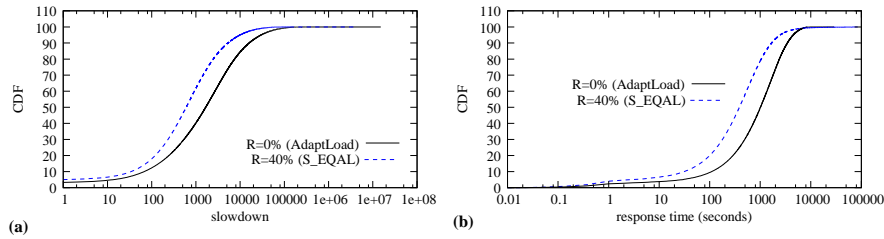
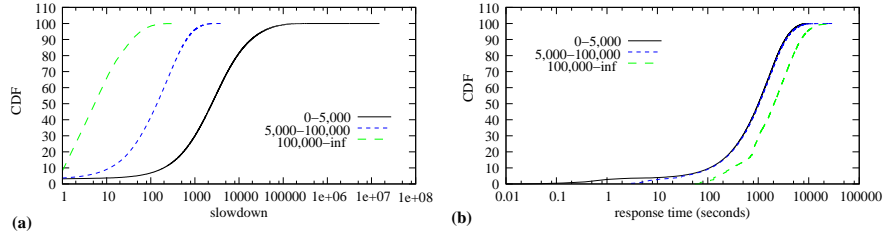


Figure 10: The CDFs of (a) slowdown and (b) response time under  $ACF_1$  inter-arrival times, with  $R = 0\%$  (ADAPTLoad) and  $40\%$ .

We also evaluate the cumulative distribution functions (CDFs) of slowdown and response time to better understand how S\_EQAL works. Figure 10 gives the CDF of slowdown and response time for all jobs (i.e., 10 million in each experiment). Since a good initial shifting value  $R$  under  $ACF_1$  inter-arrival times is equal to 40% (see Figure 8), we compare the CDFs of slowdown and response time under S\_EQAL for  $R = 40\%$  with those of ADAPTLoad (i.e.,  $R = 0\%$ ). With S\_EQAL, at least 60% of the jobs have slowdown less than 1000, this percentage reduces to 38%

(I) S\_EQAL with R = 0% (AdaptLoad)



(II) S\_EQAL with R=40%

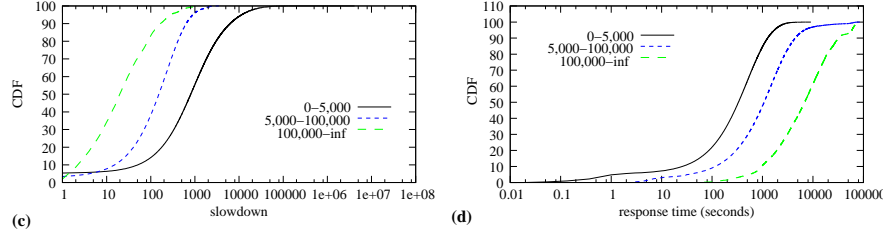


Figure 11: CDFs of slowdown and response times for three request size ranges: small [0 - 5,000), medium [5,000 - 100,000), and large [100,000 - infinity), under  $ACF_1$  inter-arrival times.

with ADAPTLLOAD, see Figure 10 (a). Figure 10 (b) shows that with ADAPTLLOAD only 50% of jobs have response time less than 1000 seconds while with S\_EQAL this percentage increases to 79%. Moreover, the figure also shows that S\_EQAL with  $R = 40\%$  makes the tail of slowdown about one order of magnitude shorter than ADAPTLLOAD. This happens, because S\_EQAL focuses on improving the performance of small jobs at the expense of the performance of large jobs, by loading heavily the servers that serve large jobs.

We present the performance of S\_EQAL for three classes of job sizes: *small* jobs that access files less than 5,000 bytes, *medium* jobs that access files with size between 5,000 and 100,000 bytes, and *large* jobs that access files with size greater than 100,000 bytes. For the specific workload that drives our simulations, *small*, *medium*, and *large* jobs represent, respectively, 84.5%, 15.4%, and 0.1% of total jobs. Per class CDFs of response times and slowdowns are illustrated in Figure 11. With ADAPTLLOAD, the performance of small and medium jobs suffers in comparison to S\_EQAL confirming our speculation that slowdown and response time tails are dominated by the (bad) performance of small jobs. This phenomenon reduces considerably with S\_EQAL: after boundary shifting the tails of slowdowns and response times are dominated by the deteriorated performance of large jobs.

### 4.1.2 Strongly correlated arrival process, $ACF_2$

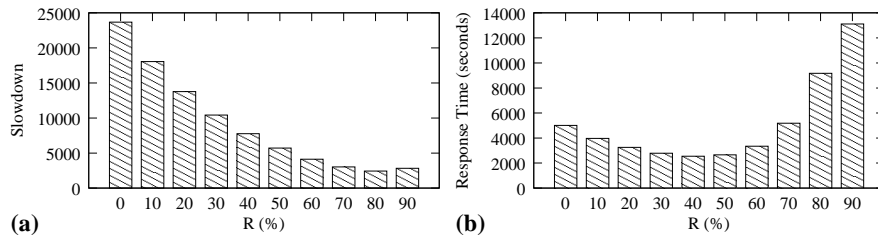


Figure 12: Average slowdown and average response time as a function of initial shifting value  $R$  with  $ACF_2$  inter-arrival times.

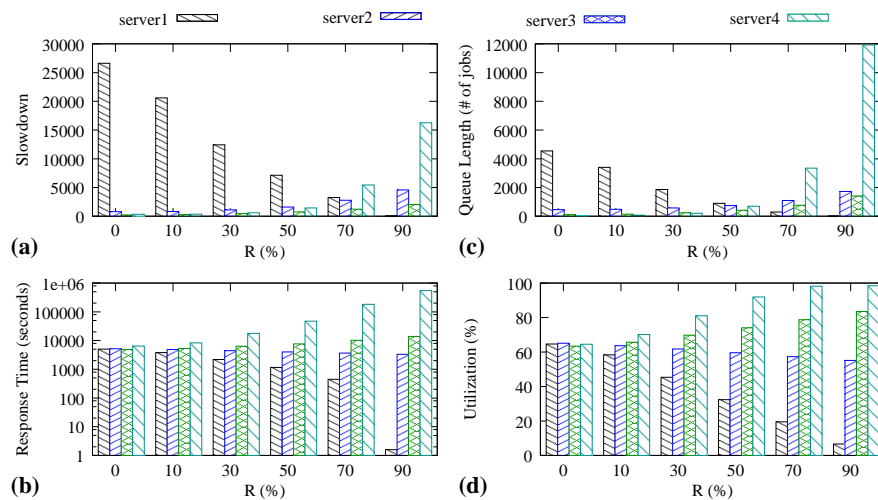


Figure 13: Per server performance metrics as a function of the initial shifting value  $R$  under  $ACF_2$  traffic: (a) average slowdown, (b) average response time, (c) average queue length and (d) average utilization. The order of bars for each policy reflects the server identity.

In this experiment, we evaluate the performance of S\_EQAL under the  $ACF_2$  arrival process. Figure 12 gives the average job slowdown and the average job response time as a function of  $R$ . In Figure 12, we observe the same trends as in Figure 8, but higher absolute performance values than under the  $ACF_1$  experiment. Compared to ADAPTLLOAD (i.e.,  $R = 0\%$ ), S\_EQAL with  $R = 40\%$  improves average response time by 49.2% and average slowdown by 67.2%.

Figure 13 illustrates per-server performance measures under  $ACF_2$  traffic. Although performance trends are similar to the  $ACF_1$  case, they are more exaggerated. Both average slowdown

and average response time of the first server reduce as  $R$  increases (see Figures 13(a)-(b)), but a turning point exists where shifting more work to subsequent servers adversely affects slowdown. CDF graphs, both across all files and for small, medium, and large ranges confirm that, as in the  $ACF_1$  experiment, the tails of the performance measures are now dominated by the deteriorated performance of large files due to shifting. CDF trends are the same as in the  $ACF_1$  experiment and are not reported here due to lack of space.

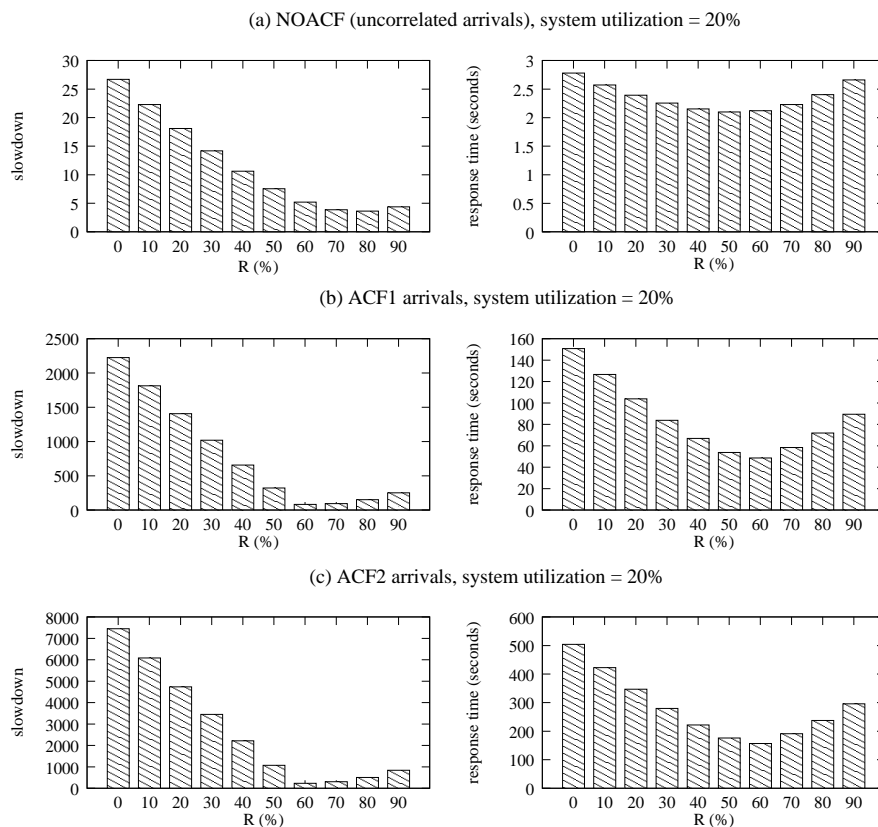


Figure 14: Average slowdown and average response time as a function of initial shifting value  $R$  when inter-arrivals are (a) uncorrelated (NOACF), (b) having  $ACF_1$  dependence, and (c) having  $ACF_2$  dependence. The average system utilization is about 20%.

### 4.1.3 Various system utilizations

In the previous sections, we evaluated the performance of S\_EQUAL for different values of  $R$  in a system with average utilization equal to about 62% and found that  $R = 40%$  is a good shifting

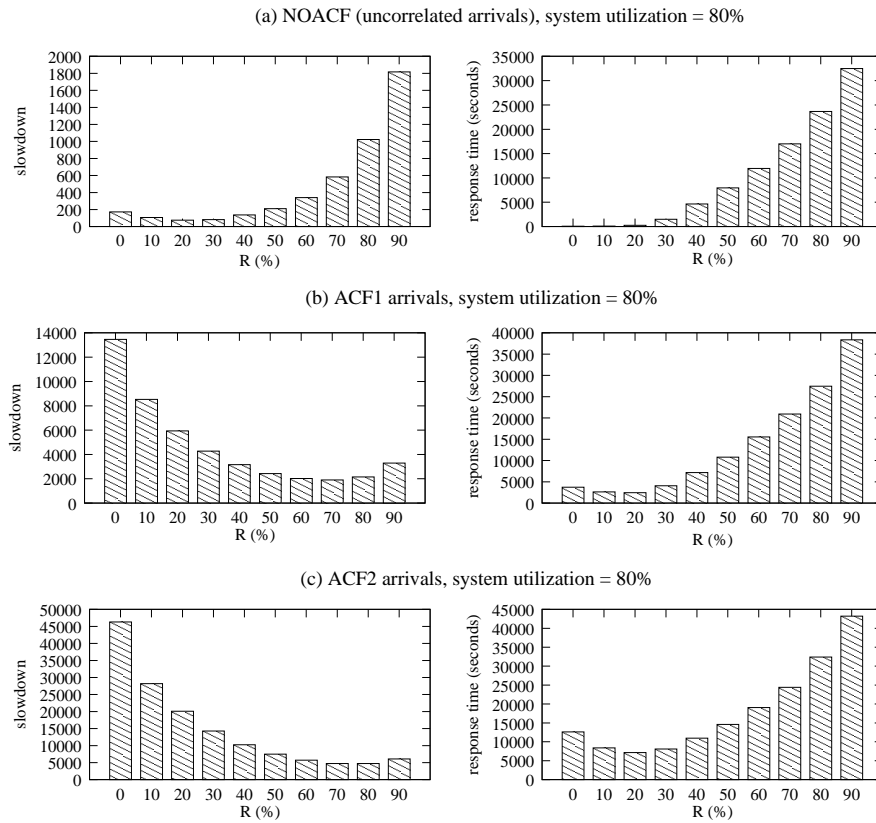


Figure 15: Average slowdown and average response time as a function of initial shifting value  $R$  when inter-arrival times are (a) uncorrelated (NOACF), (b) having  $ACF_1$  dependence, and (c) having  $ACF_2$  dependence. The average system utilization is about 80%.

value for both  $ACF_1$  and  $ACF_2$  experiments. Here, we evaluate S\_EQAL performance for different values of  $R$  under lightly loaded and heavily loaded systems.

We use the same arrival processes and the same service process as in the previous subsection, but we scale the service times to examine system performance under different utilization levels. Figure 14 illustrates the average request slowdown and the average request response time as a function of  $R$  in a system with average utilization equal to 20%. More detailed analysis shows that under uncorrelated arrivals and low system utilization levels ADAPTLLOAD does not always balance load well. This phenomenon is also identified in [25]. In this case, a shifting constant of  $R = 50\%$  corrects this known weakness of ADAPTLLOAD. Figure 14 shows that for both  $ACF_1$  and  $ACF_2$  arrivals, the best slowdown and response time is achieved with  $R = 60\%$ , making it an excellent initial shifting value in a lightly loaded system.

We also evaluate performance in a system with average utilization of 80%, see Figure 15. Under uncorrelated arrivals, S\_EQAL with  $R = 20\%$  optimizes slowdown but the best response time is achieved by ADAPTLLOAD. For both  $ACF_1$  and  $ACF_2$ , although the slowdown decreases as  $R$  increases up to  $R = 70\%$ , the lowest response time is achieved for  $R = 20\%$ . This happens because for  $R \geq 30\%$ , load imbalancing in the cluster becomes extreme, with the last server operating nearly in full capacity, i.e., reaches nearly 100% utilization. This dramatically increases response times of large jobs and, consequently, overall average job response time. We conclude that the value of the initial shifting value  $R$  should decrease as the load in the system increases.

## 4.2 D\_EQAL: On-line Policy

In the previous section, we gave a proof of concept of the performance benefits of load unbalancing. We also showed that performance improvements depend on the degree of load unbalancing determined by the initial shifting value  $R$ . A good choice of  $R$  can result in significant gains, but an unfortunate choice may also result in poor performance. Here, we present D\_EQAL, an on-line version of S\_EQAL, that continuously monitors the workload such that the effectiveness of load unbalancing becomes independent of  $R$ .

D\_EQAL continuously monitors  $C$  requests that have been just served by the cluster and re-adjusts the degree of load unbalancing on-the-fly, aiming at improving *both* average response time and average slowdown.  $C$  must be large enough to allow for performance measures to be statistically significant but also small enough to allow for quick adaptation to transient workload conditions. In the experiments presented here,  $C$  is set to 300,000. We examined the robustness of D\_EQAL with different  $C$  values ranging from 100,000 to 1,000,000 and concluded that small values of  $C$  (around 100,000) are not as effective as larger values of  $C$  (more than 200,000). This is an expected result given that  $C$  should capture changes in burstiness behavior of the process.

D\_EQAL starts by setting  $R$  to zero, i.e., no load shifting is proposed beyond the computed ADAPTLLOAD intervals. For every batch of  $C$  requests, we compare the relative performance improvement/decline of average slowdown ( $Avg_{sl\bar{d}}$ ) and average normalized response time ( $Avg_{nres}$ ) in comparison to the previous batch of  $C$  requests. The average normalized response time ( $Avg_{nres}$ )

is defined as follows

$$Avg_{nres}(k) = \frac{\text{average response time in the } k^{th} \text{ batch}}{\text{average service time in the } k^{th} \text{ batch}}$$

and aims at comparing fairly the average response times in two consecutive batches. This is particularly critical if the per-batch average service times differ significantly.

For every  $C$  requests,  $R$ , S\_EQAL’s initial shifting value, is adjusted by a fixed number  $0 < D < 100\%$  and interval boundaries are recalculated correspondingly. Figure 16 presents the D\_EQAL algorithm that implements a dynamic adjustment of  $R$  as a function of system performance measures. For the first batch of  $C$  requests,  $R = 0$ , i.e., all servers are equally loaded and the system operates using ADAPTLLOAD. This is necessary to obtain base-case performance measures. For the second batch of  $C$  requests,  $R = D$ , i.e., D\_EQAL starts exploring the performance effects of boundary shifting by decreasing the load of the first servers. In subsequent steps, average slowdown and average normalized response time of the last  $C$  requests are compared to the batch of the penultimate  $C$  requests. If system performance improves, then boundary shifting is done in the same direction, i.e., if the last adjustment shifted the work from the small (large) servers to large (small) servers, then we continue to shift more work from the small (large) servers to large (small) servers. Otherwise,  $R$  is adjusted such that boundaries are shifted in the reverse direction (see **3.b.III**, in Figure 16). In our experiments,  $D$  is set to 10%.<sup>5</sup>

Step **2.b.II** of Figure 16 provides an additional condition to avoiding over-shifting. This condition is deduced from the performance analysis of S\_EQAL (see the previous section) which shows that overloading servers 3 and 4 that serve requests for large files may significantly deteriorate average response time while maintaining acceptable slowdown. Figure 17 illustrates this behavior by plotting the average slowdown and average response time as a transient measure (i.e., across time) of every 10,000 requests. If  $R$  is set appropriately, then average slowdown and average response times change with similar rate (see Figure 17(a)). An extremely large  $R$  (i.e., load over-shifting) makes response time to increase much faster than slowdown (see Figure 17(b)). As a result, the

---

<sup>5</sup>Finding the ideal value of  $D$  is beyond the scope of this paper. The values used here are based on the experimental analysis of S\_EQAL for this specific workload. A large value of  $D$  results in faster load “unbalancing” while a smaller value results to the opposite. Dynamically adjusting the value of  $D$  using a feedback mechanism could make the policy more robust and is subject of future work.



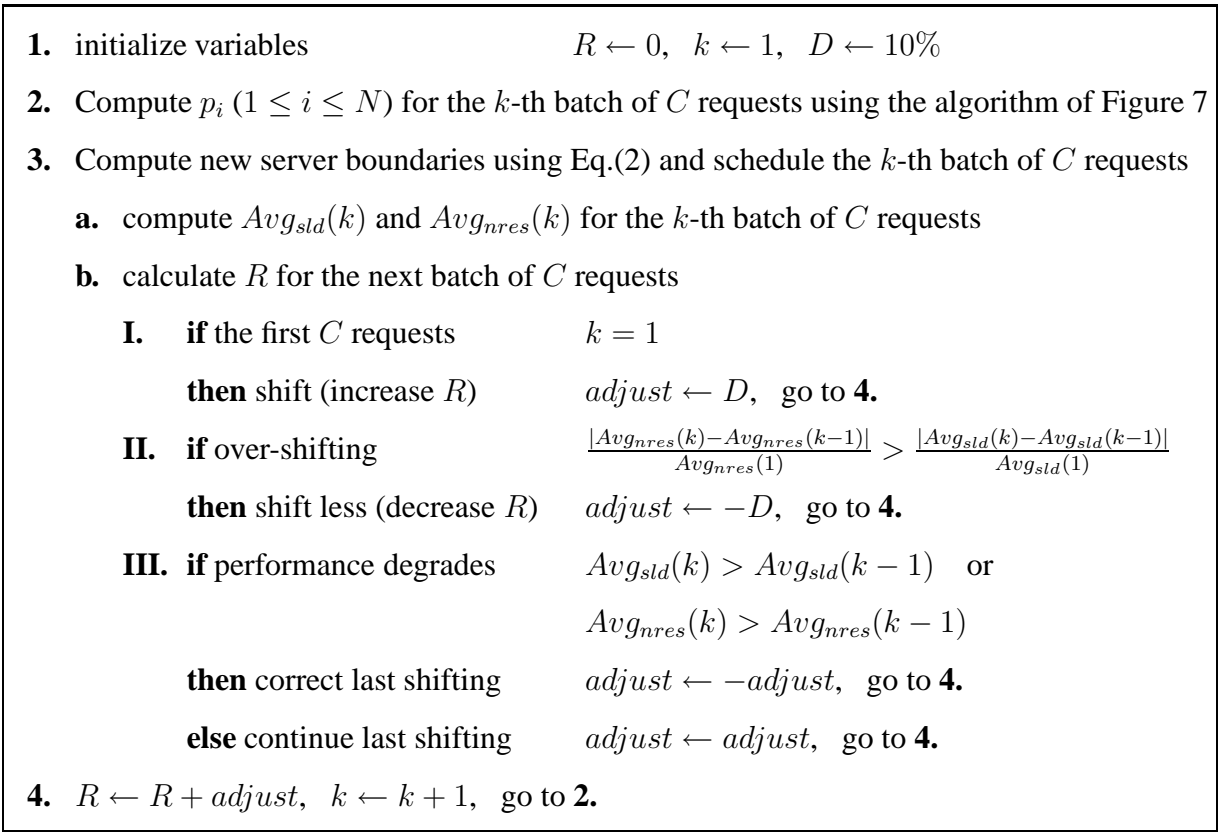


Figure 16: D\_EQAL dynamically adjusts  $R$ .

comparison of slowdown and response time provides a good indication for over-shifting. Since slowdown and response time are performance measures of different scales, we observe changes in two consecutive batches (i.e.,  $|Avg_{slid}(k) - Avg_{slid}(k-1)|$  and  $|Avg_{nres}(k) - Avg_{nres}(k-1)|$ ) and normalize them by their respective values of the first batch of  $C$  requests when no shifting occurs, (i.e.,  $Avg_{slid}(1)$  and  $Avg_{nres}(1)$  with  $R = 0$ , the original ADAPTLOAD ) for a fair comparison.

#### 4.2.1 Performance of D\_EQAL

In this section, we evaluate the effectiveness of D\_EQAL. We compare ADAPTLOAD, i.e., S\_EQAL with  $R = 0\%$ , S\_EQAL with various values of its initial shifting value  $R$ , and D\_EQAL. Note that D\_EQAL starts with  $R = 0$ , which implies that we rely on the algorithm to find the appropriate  $R$ . Results for various system utilization levels (i.e., 20%, 62%, and 80%) are presented in Figures 18-20. In all graphs, D\_EQAL is comparable to the best performing S\_EQAL. D\_EQAL manages to

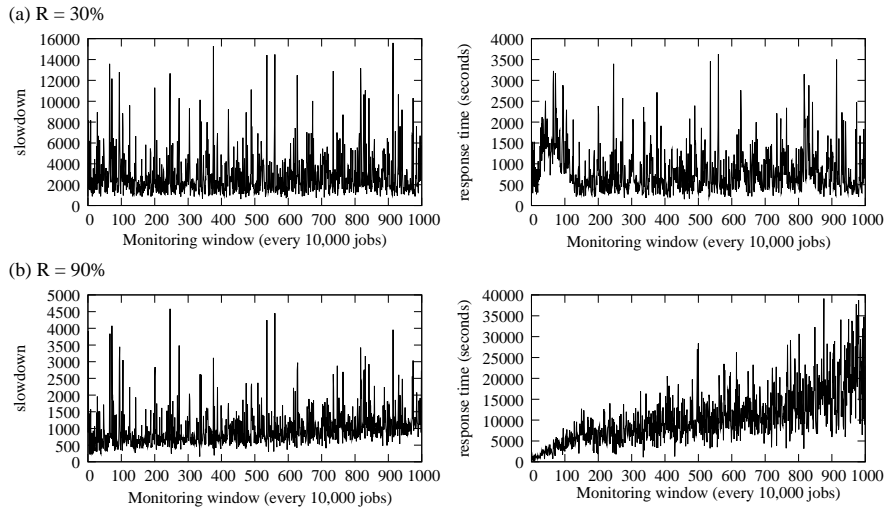


Figure 17: The average slowdown and average response time of every 10,000 requests under  $ACF_1$  arrivals when (a)  $R = 30\%$  and (b)  $R = 90\%$ . The system utilization is 62%.

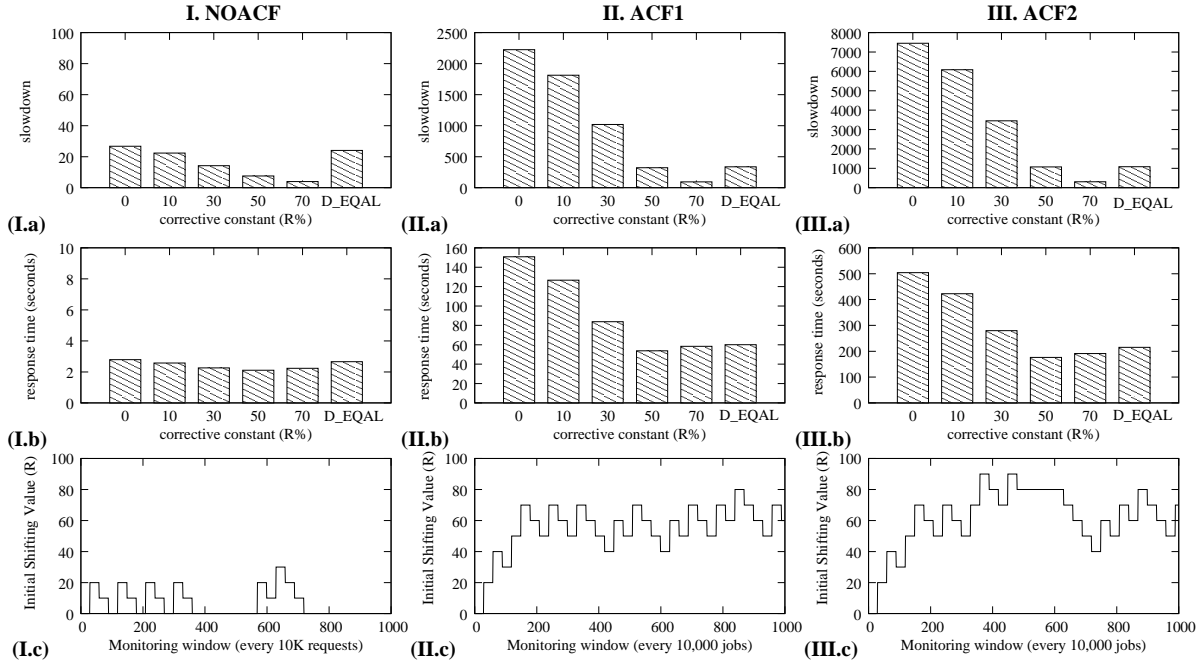


Figure 18: Performance effects of autocorrelation under low system utilization (20%). The first two rows give the average slowdown and the average response time. The third row shows how the initial shifting value  $R$  is updated as a function of time (measured in processed requests) for  $C = 300,000$ .

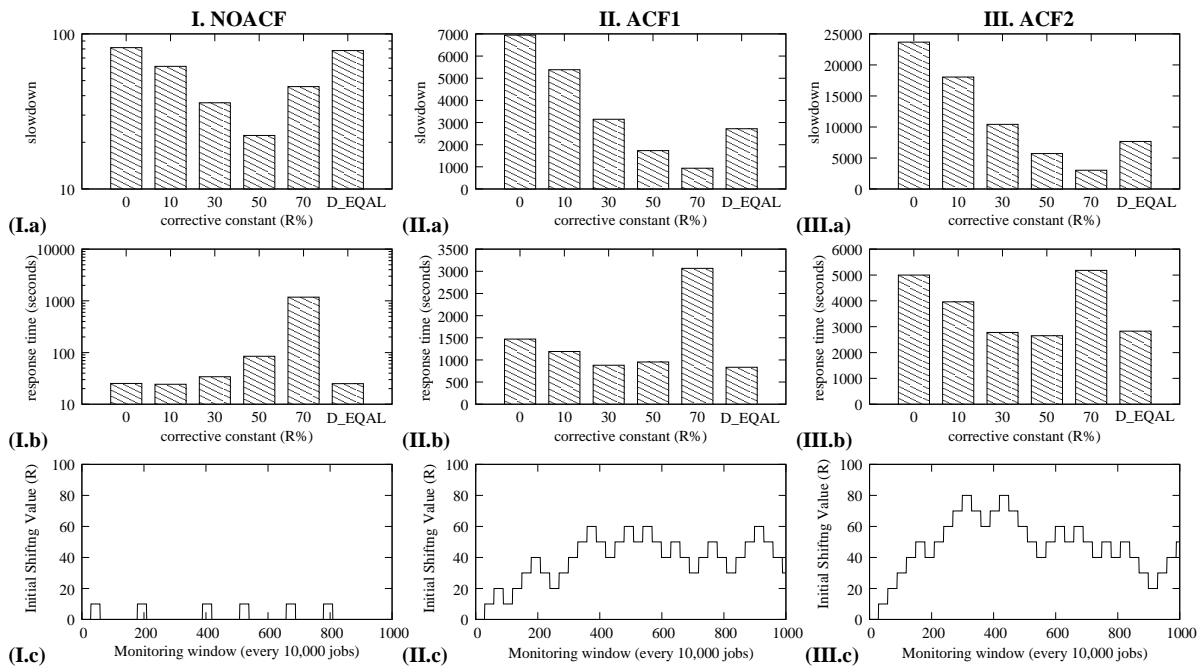


Figure 19: Performance effects of autocorrelation under medium system utilization (62% utilization level). The first two rows give the average slowdown and the average response time. The third row shows how the initial shifting value  $R$  is updated as a function of time (measured in processed requests) for  $C = 300,000$ .

adjust  $R$  such that both slowdown (see Figures 18-20(a)) and response times (see Figures 18-20(b)) are improved.

Figures 18-20(c) show how the algorithm changes  $R$  throughout the duration of the experiment. With no autocorrelation in the arrival stream,  $R$  almost always remains equal to 0, irrespective of the system utilization level, essentially the policy behaves like ADAPTLOAD. With  $ACF_1$  or  $ACF_2$  arrivals,  $R$  converges toward the best performing static value as seen in the analysis of the performance of S\_EQUAL, see Section 4.1.3.

## 5 Conclusions

We presented evidence via detailed simulations that size-based policies for load balancing in homogeneous clusters become ineffective when the arrival process is autocorrelated. If the arrival

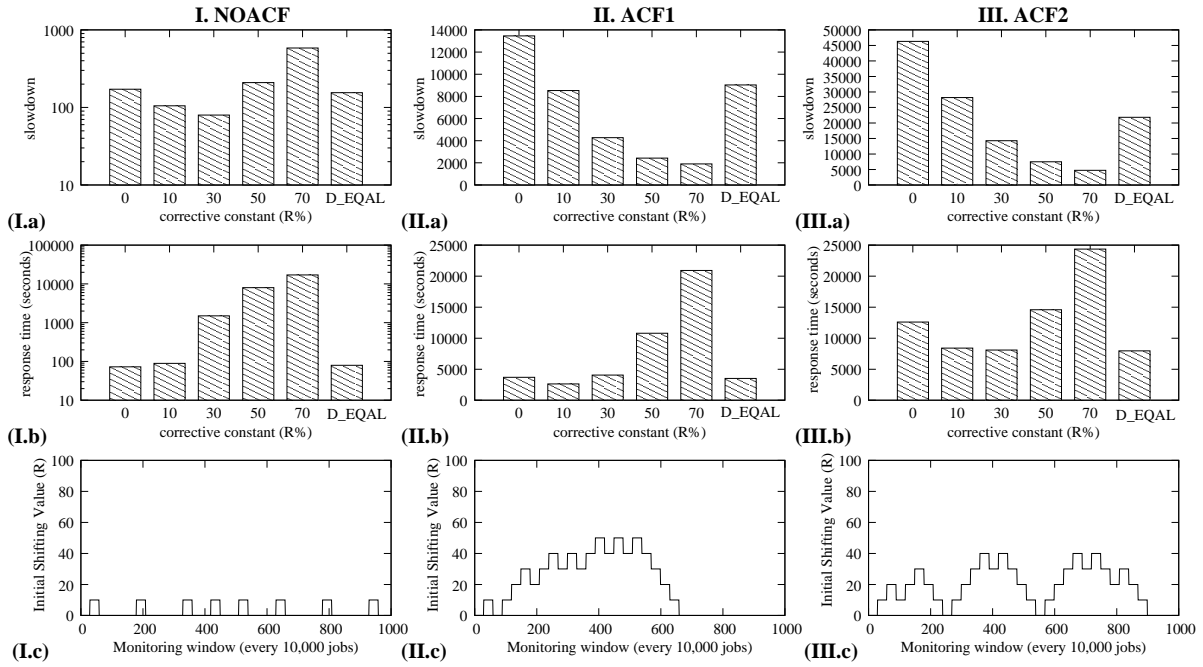


Figure 20: Performance effects of autocorrelation under high system utilization (80% utilization level). The first two rows give the average slowdown and the average response time. The third row shows how the initial shifting value  $R$  is updated as a function of time (measured in processed requests) for  $C = 300,000$ .

process is autocorrelated, then the basic premise of size-based policies, i.e., balancing the load by keeping each server equally utilized while serving jobs of similar size in each server, may actually hurt performance as per-server performance is sensitive to not only its utilization level but most importantly to the dependence structure in the arrival stream of jobs that it serves.

We propose a new size-based load balancing policy, called D\_EQAL, that still strives to serve jobs of similar size in each server but per-server utilization levels depend on the autocorrelation of the arrival process to that particular server. As a result of this effort, if there is autocorrelation in the arrival stream to the cluster, all servers may not be equally utilized (i.e., load in the system becomes unbalanced) but this imbalance results in significant performance benefits. If there is no autocorrelation in the arrival stream, then D\_EQAL seamlessly balances load across all servers as ADAPTLOAD does (i.e., it behaves like a typical size-based policy). D\_EQAL does not require any prior knowledge of the correlation structure of the arrival stream or of the job size distribution.

Using detailed simulations we show that D\_EQAL can be used on-line: by monitoring system performance measures, it self-adjusts its configuration parameters to transient workload conditions and significantly improves performance under correlated arrivals.

## Acknowledgments

We thank the anonymous reviewers whose comments significantly improved this presentation. This work was partially supported by the National Science Foundation under grants ITR-0428330 and CNS-0720699, and by Seagate Research. A preliminary version of this paper appeared at the 26<sup>th</sup> International Conference on Distributed Computer Systems, (ICDCS'06), Lisboa, Portugal, July 2006 [24].

## References

- [1] D. Abendroth and U. Killat. Intelligent Shaping: Well Shaped Throughout the Entire Network? In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, volume 2, pages 912–919, New York City, NY, June 2002.
- [2] A. M. Adas and A. Mukherjee. On Resource Management and QoS Guarantees for Long Range Dependent Traffic. In *Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM 1995)*, volume 2, pages 779–787, Boston, MA, April 1995.
- [3] M. Andreolini, M. Colajanni, and R. Morselli. Performance Study of Dispatching Algorithms in Multi-tier Web Architectures. *ACM SIGMETRICS Performance Evaluation Review*, 30(2):10–20, September 2002.
- [4] M. Arlitt and T. Jin. Workload Characterization of the 1998 World Cup Web Site. Technical Report HPL-1999-35R1, Hewlett-Packard Laboratories, September 1999.
- [5] D. Bushmitch, S. S. Panwar, and A. Pal. Thinning, Striping and Shuffling: Traffic Shaping and Transport Techniques for Variable Bit Rate Video. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 2002)*, volume 2, pages 1485–1491, Taipei, November 2002.
- [6] L. Cherkasova, W. Tang, and S. Singhal. An SLA-Oriented Capacity Planning Tool for Streaming Media Services. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN 2004)*, pages 743–752, Florence, Italy, June 2004.
- [7] K. J. Christensen and V. Ballingam. Reduction of Self-Similarity by Application-Level Traffic Shaping. In *Proceedings of the 22nd Annual IEEE Conference on Local Computer Networks (LCN 1997)*, pages 511–518, Minneapolis, MA, November 1997.
- [8] A. Erramilli, O. Narayan, and W. Willinger. Experimental Queueing Analysis with Long-Range Dependent Packet Traffic. *IEEE/ACM Transactions on Networking*, 4(2):209–223, April 1996.
- [9] H. Feng, M. Visra, and D. Rubenstein. Optimal State-Free, Size-aware Dispatching for Heterogeneous M/G/-type Systems. *Performance Evaluation Journal*, 62(1-4):475–492, November 2005.
- [10] M. Harchol-Balter, M. Crovella, and C. D. Murta. On Choosing a Task Assignment Policy for a Distributed Server System. *Journal of Parallel and Distributed Computing*, 59(2):204–228, November 1999.

- [11] M. Harchol-Balter and A. Downey. Exploiting Process Lifetime Distributions for Dynamic Load Balancing. *ACM Transactions on Computer Systems*, 15(3):253–285, August 1997.
- [12] D. Heyman and D. Lucantoni. Modeling Multiple IP Traffic Streams with Rate Limits. *IEEE/ACM Transactions on Networking*, 11(6):948–958, December 2003.
- [13] L. Kleinrock. *Queueing Systems, Volume I: Theory*. Wiley, 1975.
- [14] N. Mi, Q. Zhang, A. Riska, and E. Smirni. Performance Impacts of Autocorrelation in TPC-W. In *Proceedings of the 26th International Symposium on Computer Performance, Modeling, Measurements, and Evaluation (Performance 2007)*, Cologne, Germany, October 2007.
- [15] 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 *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, pages 205–216, San Jose, CA, October 1998.
- [16] A. Riska and E. Riedel. Disk Drive Level Workload Characterization. In *Proceedings of the USENIX Annual Technical Conference*, pages 97–102, Boston, MA, June 2006.
- [17] A. Riska and E. Riedel. Long-Range Dependence at the Disk Drive Level. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems (QEST 2006)*, pages 41–50, September 2006.
- [18] Y. M. Teo and R. Ayani. Comparison of Load Balancing Strategies on Cluster-based Web Servers. *Transactions of the Society for Modeling and Simulation*, 77(5-6):185–195, November 2001.
- [19] V. Ungureanu, B. Melamed, P. G. Bradford, and M. Katehakis. Class-Dependent Assignment in Cluster-based Servers. In *Proceedings of the 2004 ACM symposium on Applied computing (SAC 2004)*, pages 1420–1425, Nicosia, Cyprus, March 2004.
- [20] U. Vallamsetty, K. Kant, and P. Mohapatra. Characterization of E-Commerce Traffic. In *Proceedings of the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2002)*, pages 137–144, Newport Beach, California, 2002.
- [21] F. Xue and S. J. B. Yoo. Self-similar Traffic Shaping at the Edge Router in Optical Packet-switched Networks. In *Proceedings of IEEE International Conference on Communications (ICC 2002)*, volume 4, pages 2449–2453, April 2002.
- [22] Q. Zhang, L. Cherkasova, and E. Smirni. FlexSplit: A Workload-Aware, Adaptive Load Balancing Strategy for Media Clusters. In *Proceedings of Multimedia Computing and Networking (MMCN’06)*, San Jose, CA, January 2006.
- [23] Q. Zhang, A. Heindl, and E. Smirni. Characterizing the BMAP/MAP/1 Departure Process via the ETAQA Truncation. *Stochastic Models*, 21(2-3):821–846, 2005.
- [24] Q. Zhang, N. Mi, A. Riska, and E. Smirni. Load Unbalancing to Improve Performance under Autocorrelated Traffic. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006)*, Lisboa, Portugal, June 2006.
- [25] Q. Zhang, A. Riska, W. Sun, E. Smirni, and G. Ciardo. Workload-Aware Load Balancing for Clustered Web Servers. *IEEE Transactions on Parallel and Distributed Systems*, 16(3):219–233, March 2005.