

Load Unbalancing to Improve Performance under Autocorrelated Traffic*

Qi Zhang Ningfang Mi
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
{qizhang,ningfang}@cs.wm.edu

Alma Riska
Seagate Research
1251 Waterfront Place
Pittsburgh, PA 15222
alma.riska@seagate.com

Evgenia Smirni
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
esmirni@cs.wm.edu

Abstract

Size-based policies have been shown to successfully balance load and improve performance in homogeneous cluster environments where a dispatcher assigns a job to a server strictly based on the job size. While the success of size-based policies is based on separating jobs to different servers according to their sizes by avoiding the unfavorable performance effects of having short jobs been stuck behind long jobs, we show that their effectiveness quickly deteriorates in the presence of job arrivals that are characterized by correlation in their dependence structure. We propose a new policy that still strives to separate jobs to servers according to their sizes, but this separation is biased by the effort to reduce the performance loss due to autocorrelation in the streams of jobs that are directed to each server. As a result of this effort, not all servers are equally utilized (i.e., the load in the system becomes unbalanced) but the performance benefits of this load unbalancing are significant. The proposed policy can be used on-line, i.e., it does not assume any knowledge neither of the correlation structure of the arrival stream, nor of the job size distribution in the system. Via detailed trace-driven simulation we quantify the performance benefits of the proposed policy and we show that it can effectively self adjust its configuration parameters to improve performance under continuously changing workload conditions.

Keywords: load balancing, autocorrelated arrivals, highly variable service times, self adaptive policies.

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

*This work was partially supported by the National Science Foundation under grants CCR-0098278, ACI-0090221, and ITR-0428330, and by Seagate Research.

set of homogeneous hosts behaves as a single host. Jobs (or requests) arrive at a dispatcher which then forwards them to the appropriate server.¹ While there exists no central waiting queue at the dispatcher, each server has a separate queue for waiting jobs and a separate processor, see Figure 1. The dispatching policy is critical for system performance and strongly depends on the stochastic characteristics of the jobs that request service as well as on the performance measures that the system strives to optimize.

Prior research has shown that the job service time distribution is critical for the performance of load balancing policies in such a setting [10, 9]. If job service times are highly variable, including job service times that are best characterized using heavy-tailed distributions, then policies that balance the load in the system by using *only* the size of each incoming job to determine the server that will be dispatched to, have been shown optimal if the performance goal is to minimize the expected job completion time, job waiting time, and job slowdown [7, 19]. Several types of clustered systems can take advantage of size-based policies, e.g., locally-distributed Web server cluster architectures [2, 19], content-distribution networks and media-server clusters [18, 4], and large storage systems.

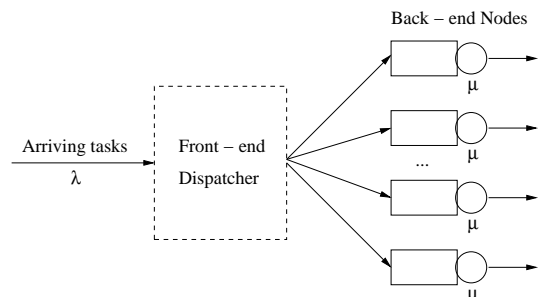


Figure 1. Model of a clustered server.

Nonetheless, size-based solutions are not adequate if the

¹In this paper we are using the terms “jobs” and “requests” interchangeably.

arrival streams in the dispatcher are *autocorrelated*. Indeed, conventional wisdom has it that the arrival process in Internet servers is not independent and it is an effect of the self-similar nature of the network traffic [16]. Furthermore, autocorrelated flows in the arrival process has been observed in systems including multi-tiered systems [13], large storage systems [8], an effect that has been shown to be detrimental for performance [6]. To alleviate the negative effects of autocorrelation, traffic shaping has been used by dropping, reordering, or delaying selected requests [17, 5, 1].

In this paper, we show that size-based load balancing policies cease to be effective if the workload arrival process is autocorrelated. We show that as autocorrelation in the arrival process increases, the performance benefits of size-based policies diminish. Based on our observations, we propose a size-based load balancing policy that aims at reducing the performance degradation due to autocorrelation in each server, while maintaining the property of similar job sizes been served by the same server. This new policy, called D_EQAL, strives to equally distribute work guided by autocorrelation and load, and effectively unbalances the load in the system: not all servers are equally utilized any more, 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 intensity of the dependence structure in the arrival streams. By observing past arrival and service characteristics as well as past performance, it self-adjusts its configuration parameters. 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 motivates our work by showing that autocorrelation and high variability characterize the workload in storage systems. In Section 3 we compare the performance of a size-based policy with several classic policies in the presence of autocorrelated arrival flows in the system. The proposed on-line size-based policy is presented in Section 4 and its performance is evaluated via simulation. Section 5 summarizes our contributions and outlines future work.

2 Motivation

In this section we present data that have been measured on a real system to confirm dependence in the arrival stream as well as high variability in the service process. We present data measured in various storage clusters [14]. In storage clusters the existence of caches at various levels contributes to highly variable inter-arrivals and service times. The cache hierarchy together with resource distribution further results in burstiness in the request arrival process.

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). 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

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

where μ is the mean and δ^2 is the common variance of $\{X_n\}$. The argument k is called the lag and denotes the time separation between the occurrences 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 series is independent, i.e., uncorrelated. In most cases ACF approaches zero as k increases. The ACF's decay rate distinguishes processes as short-range dependent (SRD) or long-range dependent (LRD).

In Figure 2(a), we present the ACF of arrivals at storage systems supporting (dedicatedly) various applications. As expected, for different applications the dependence structure of the arrivals is different and it is a result of multiple factors including the architecture of the storage system, the file system running on top of the storage system, and the I/O path hierarchy together with the resource managing policies at all levels of the I/O path. However, independently of all these factors and with only few exceptions, all measurements show that arrivals at the storage system are correlated.

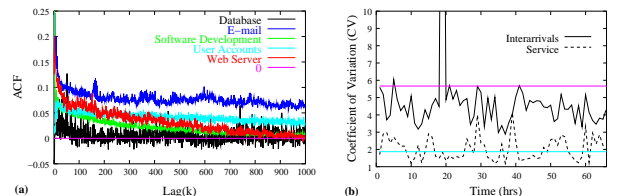


Figure 2. (a) ACF of the arrival process at the storage subsystem for various applications. (b) CV of inter-arrivals and service times at the Web server storage subsystem.

We focus now only on the storage system that supports a Web server and evaluate the variability of both request inter-arrivals and request service times. The trace data do not contain the explicit service times at the storage system but the request response time, i.e., the sum of waiting in the queue and service time for each request. Nonetheless, for the results that we report here, the system operates under very low load with essentially only one outstanding request most of the time in the system. Therefore, the request

completion time closely approximates the request service time. Note that by approximating the service times with the response times in this fashion enables us to accurately estimate the mean and CV of service times, but it does not allow accurately estimate the dependence structure of the service process. Hence, throughout the paper, we assume that the service process in each server is independent. Figure 2(b) plots the CV of inter-arrivals and service times over one hour intervals for the duration of the trace, i.e., 66 hours. Note that inter-arrivals are more variable than the service times but both processes have high variability.

Using the data from the storage system of the Web server, we parameterize a simple MMPP/H₂/1 queuing model to analyze the effect of the autocorrelation in the inter-arrival process on performance. The arrival process is drawn from a 2-stage MMPP process with mean inter-arrival time equal to 13.28 ms and CV equal to 5.67, as derived by the arrival process of the Web server presented in Figure 2(b).² The service process is drawn from a 2-stage hyperexponential (H₂) distribution with mean service time equal to 3 ms and CV equal to 1.85, respectively. Inter-arrival times are scaled so that we examine the system performance under different utilization levels. We also present experiments with different MMPPs such that we maintain the same mean and CV in the arrival process, but we change its autocorrelation structure so that there is no autocorrelation (ACF=0, for all lags), short-range dependence (SRD) with ACF starting at 0.47 at lag=1 but decaying to 0 at lag=300, and long-range dependence (LRD) with ACF starting at 0.47 at lag=1 but reaching 0.05 beyond lag=700.

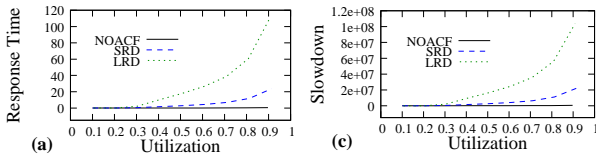


Figure 3. (a) response time, and (b) slow-down as a function of system utilization when inter-arrivals are independent (no ACF), having short range dependence (SRD), and having long-range dependence (LRD).

Figure 3 presents performance measures for the MMPP/H₂/1 queuing model as a function of system utilization. We measure performance (here and throughout the paper) by reporting on response time which is the sum of the request service time and its waiting time in the queue,

²We selected a Markovian-Modulated Poisson Process (MMPP), a special case of the Markovian Arrival Process (MAP) [12], to model autocorrelated service times because it is analytically tractable. Its basic building block is a simple exponential but it can be easily parameterized to show dependence in its structure.

queue length which is the total number of requests in the server queue including the one in service, and the request slowdown which is the ratio of the response time of a request to its service time. The effect of ACF on system performance is tremendous: the higher the ACF, the worse the system performance, which can worsen as much as 3 orders of magnitude when comparing to the case with no ACF arrivals.³ It is not only variability in the arrival and service processes that hurt performance, but more importantly the dependence structure in the arrival process. In the next section, we present performance results that show how the dependence structure in the arrival process can become critical for the performance of load balancing policies.

3 Autocorrelation Effects on Load Balancing Policies

In this section, we use trace driven simulation to examine the performance impacts of autocorrelated arrivals in load balancing policies in the simple cluster depicted in Figure 1. We assume that the number of nodes is equal to four.⁴

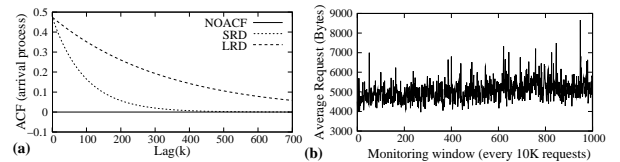


Figure 4. (a) ACF for the three arrival processes used in the simulation and (b) Average request size for every 10000 requests in the ten million sample space.

The service process is obtained from traces of the 1998 World Soccer Cup Web site,⁵ that have been used in several studies to evaluate the performance in load balancing policies in clustered web servers [19, 15]. Trace data were collected during 92 days, from 26 April 1998 to 26 July 1998, see [3] for more details. Here, 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. In the trace used for our experiments, the average size of a requested file is 5059 bytes and

³Because of the scale used in the figure and because of the difference of the three curves, the performance measures with no ACF look flat. With no ACF for utilization equal to 0.9, queue length is equal to 152 as expected, but this number is dwarfed in comparison to the LRD and SRD numbers.

⁴Experiments with larger number of nodes have been also done but results are qualitatively the same and are not reported here due to lack of space.

⁵Available from the Internet Traffic Archive at <http://ita.ee.lbl.gov>.

its coefficient of variation (CV) is 7.56. Figure 4(b) plots the average request size for batches of 10,000 requests for the duration of the trace, and shows that the average transferred size varies across time.

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 a 2-stage MMPP, which with appropriate parameterization allows for changing *only* the ACF while maintaining the same mean and CV, that are equal to 1 and 4.5, respectively. The ACF of the three arrival processes that we use here is illustrated in Figure 4.

3.1 Load Balancing Policies

We compare the performance of the following policies: ADAPTLLOAD, a size-based policy that does not require *a priori* knowledge of the service time distribution and has been shown to be effective under changing workload conditions [19], the *Join Shortest Weighted Queue (JSWQ)* policy [19], *Join Shortest Queue (JSQ)* [11], and *Round Robin (RR)*. The policies are summarized as follows:

- **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 requested file 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 CDF of the request sizes and the amount of total work is \bar{S} . By sending requests of similar sizes to each server, the policy improves average job response time and average job slowdown by avoiding having short jobs been stuck after long jobs in the queue. For a transient workload, the value of the $N - 1$ size boundaries s_1, s_2, \dots, s_{N-1} is 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 10000.

- **JSWQ:** The length of each queue in the system is weighed by the size of queued requests, therefore each incoming request is routed to 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.

3.2 Performance Analysis

Using trace-driven simulation we compare the performance of the four policies. In all our experiments, we consider a cluster of four homogeneous back-end servers that serve requests in a *first-come-first-serve (FIFO)* order.

We evaluate the effect of autocorrelated interarrival times on the performance of load balancing policies by analyzing the response time (i.e., wait time plus service time), the average queue length (i.e., the total number of jobs in the server, both waiting and in service), the average slowdown (i.e., the ratio of the actual response time of a request to its service time), and the mean utilization. Figure 5 plots performance results for the four load balancing policies in the three different experiments. The figure shows that correla-

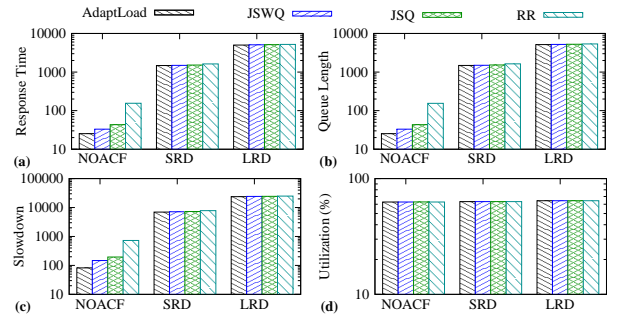


Figure 5. Performance metrics under four load balancing policies: (a) average response time, (b) average queue length, (c) average slowdown, and (d) average utilization.

tion in the arrival process degrades overall system performance for all four policies. For example the overall performance under independent arrivals (NOACF) is two orders of magnitude better than under SRD interarrivals, and three orders of magnitude better than under LRD interarrivals, despite the fact that average system utilizations are exactly the same for all experiments, i.e., the average utilizations are about 62%, see Figure 5(d).⁶ Most importantly, the figure also shows that ADAPTLLOAD outperforms all policies under independent interarrivals *only*, see Figure 5(a)-(c).

⁶Per server utilizations for all experiments remain the same, and equal to about 62%, but are not reported here due to lack of space.

Under correlated arrival processes, ADAPTLLOAD’s performance is comparable to the three other policies, essentially showing that separating requests according to their sizes is not sufficient.

4 Unbalancing Load to Improve Performance

In this section, we propose an enhancement to the ADAPTLLOAD policy that accounts for dependence in the arrival process by relaxing ADAPTLLOAD’s goal to balance the work among all nodes of the cluster. The proposed policy strives to judiciously *unbalance* the load among the nodes by moving jobs from the nodes with a strongly correlated arrival process to nodes with weaker correlation in their inter-arrival times. First we present a static version of the policy where the load of the servers with correlated interval times is reduced by a static percentage while the load of servers with no autocorrelation in their arrival process increases. Then, we present a dynamic version of the same policy where measured workload characteristics and policy performance measures guide load unbalancing in the system to improve overall system performance.

4.1 S_EQAL: Static Policy

Recall that ADAPTLLOAD is based on the idea that given that in an N -server cluster the amount of total work is \bar{S} , then the best performance is achieved if requests are assigned to the servers such that each server serves \bar{S}/N of the work, i.e., load is well balanced across all servers. Associating the request size with the work a server has to do, ADAPTLLOAD equally distributes the work among servers by determining boundaries of request sizes for each server. 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 Eq. (1).

S_EQAL uses the same histogram information, but sets the new boundaries s'_i by weighting the work assigned to each server as a function of the degree of correlation in the arrival process based on the observation that in order to achieve similar performance levels under autocorrelated arrivals, the system utilization must be lower than under independent arrivals.

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 $1 \leq i \leq N$. Note that p_i can take both negative and positive values. A negative p_i indicates that the amount of work assigned at server i should be less than the equal share of \bar{S}/N . A positive p_i indicates that the amount of work assigned at server i should be higher than the equal share of \bar{S}/N . Because the shifting

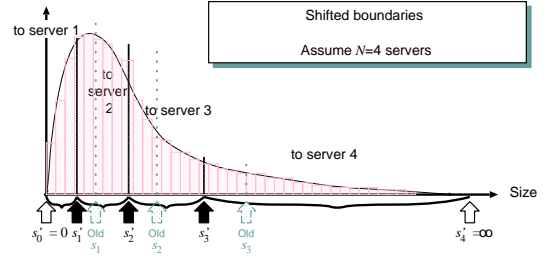


Figure 6. S_EQAL’s high level idea to recalculate boundaries under autocorrelated inter-arrival times.

percentage p_i simply shifts the amount of work from one server to another it should satisfy the equation $\sum_{i=1}^N p_i = 0$ 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.

First, we statically define the values of p_i for $1 \leq i \leq N$, by letting p_1 be equal to a pre-determined corrective constant R , $0\% \leq R < 100\%$, and then by calculating the rest of the shifting percentages p_i for $2 \leq i \leq N$ using a semi-geometric increasing method, as described by the algorithm in Figure 7. Because the first server is usually the one that serves the small requests and has strong autocorrelated inter-arrival times, the shifting percentage p_1 is negative, i.e., $p_1 = -R$. For example, if we define $R = 10\%$ then the shifting percentages for a 4-server cluster are $p_1 = -10\%$, $p_2 = -1.67\%$, $p_3 = 3.33\%$ and $p_4 = 8.34\%$. For $R = 20\%$ the shifting percentages are twice as high as in the case of $R = 10\%$. i.e., $p_1 = -20\%$, $p_2 = -3.34\%$, $p_3 = 6.67\%$, and $p_4 = 16.67\%$.

4.1.1 Arrival process with short/long-range dependence

We evaluate the performance of S_EQAL using the short range dependent arrival process used in Section 3. First, we quantify the effect of the corrective constant R that we use to generate the values of the shifting percentages p_i for $1 \leq i \leq N$ by computing the average slowdown and average response time of requests under S_EQAL for different values of R . We present our findings in Figure 8. $R = 0\%$ corresponds to the the original ADAPTLLOAD, i.e., no shifting of boundaries. Figure 8(a) shows that the average slowdown of all requests improves as R increases (i.e., the boundaries are shifted to the left compared to the original ADAPTLLOAD). We observe that the best performance

1. initialize variables
 - a. initialize a variable: $adjust \leftarrow -R$
 - b. initialize the shifting percentages:

$$p_i \leftarrow 0 \text{ for all } 1 \leq i \leq N$$
2. for $i = 1$ to $N - 1$ do
 - a. add $adjust$ to p_i : $p_i \leftarrow p_i + adjust$
 - b. for $j = i + 1$ to N do
 - equally distribute $adjust$ to the remaining servers:

$$p_j \leftarrow p_j - \frac{adjust}{N-i}$$
 - c. reduce $adjust$ to half: $adjust \leftarrow adjust/2$

Figure 7. Setting the shifting percentages p_i for S_EQUAL.

is achieved for $R = 80\%$ (i.e., $p_1 = -80\%$). However,

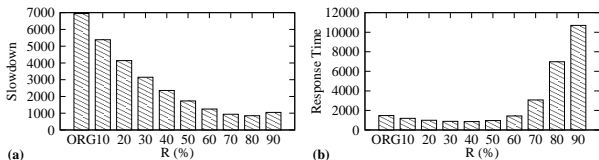


Figure 8. Average slowdown and average response time as a function of the corrective constant R under SRD inter-arrival times.

Figure 8(b) indicates that the best performance for response time is achieved when $R = 40\%$. Therefore, a good corrective constant is $R = 40\%$, where average slowdown improves by 75.1%. Average response time improves by 41.9% when compared to the original ADAPTLLOAD.

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 utilization of the first two servers decrease while the 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 do not increase their slowdown significantly for small values of R . For $R = 90\%$, slowdown at server 4 becomes very high, almost twice as high as for server 1 under the original ADAPTLLOAD. The average per-server queue length behaves similarly to the 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 performance values

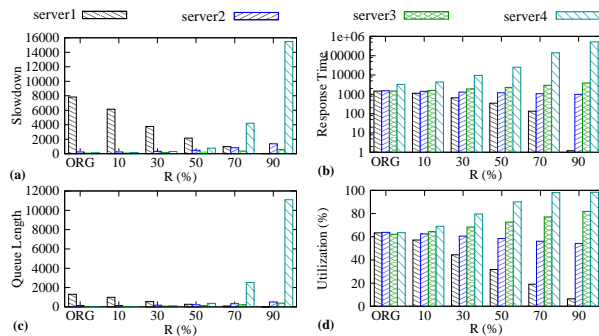


Figure 9. Per server performance measures: (a) average slowdown, (b) average response time, (c) average queue length and (d) average utilization as a function of the corrective constant R with SRD inter-arrival times. The order of bars for each policy reflect the server identity.

to the overall performance degradation is not significant. As R increases, more jobs are assigned to higher servers, which counterbalances the benefits of decreased utilization at the first servers.

The performance results of the long range dependent arrival are not presented here due to limited space. But we find that the performance trends are similar as with the SRD case though they are more exaggerated.

4.2 D_EQUAL: On-line Policy

In the previous section we gave a first proof of concept that load imbalancing can be beneficial for performance in clusters with autocorrelated inter-arrival times and heavy tailed service requests, but performance improvements depend on the degree of load imbalancing that is introduced by the corrective constant R . A good choice of R can result in significant performance improvements, but an unfortunate choice may also result in poor performance. Here we present an on-line version of the policy that monitors the workload as well as the effectiveness of load balancing, and its performance is now independent of the choice of R . Based on continuous monitoring, the policy readjusts the degree of load imbalancing on-the-fly while aiming at improving *both* average response time and average slowdown.

We use an updating window of C requests that have been served by the cluster. C must be large enough to allow for statistically significant performance measurement but also small enough to allow for quick adaptation to transient workload conditions. In the experiments presented here C is set to 300K. The policy starts by setting R to zero, i.e., no load shifting is proposed beyond the computed ADAPT-

```

1. initialize
  a. set  $R \leftarrow 0$ 
  b.  $k \leftarrow 0$ 
2. every  $C$  requests
  a. compute the current performance metrics  $Avg_{std}(k)$ 
    and  $Avg_{nres}(k)$ 
  b. if  $(k = 0)$ 
    then I. Correct left
        II. go to 3.
  c. if  $\frac{Avg_{nres}(k) - Avg_{nres}(k-1)}{Avg_{nres}(0)} > \frac{Avg_{std}(k) - Avg_{std}(k-1)}{Avg_{std}(0)}$ 
    then I. Correct right
        II. go to 3.
  d. if  $(Avg_{std}(k) > Avg_{std}(k-1)$  or
     $Avg_{nres}(k) > Avg_{nres}(k-1))$ 
    then Correct reversely
    else Correct continuously
3. Compute  $p_i$  using the algorithm of Figure 7
4.  $k \leftarrow k + 1$ 
5. goto 2.

```

Figure 10. D_EQAL: dynamically adjusting R .

LOAD intervals. For every batch of C requests, we compare the relative performance improvement/decline in comparison to the previous batch of C requests.⁷ The two performance measures that we examine are the average slowdown (Avg_{std}) and the average normalized response time (Avg_{nres}), which is defined as follows:

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

Then, according to the comparison of the values of average slowdown and normalized response times, we readjust R by a *small* value adj , which in our experiments is set to 10% (i.e., 10% of the load is shifted left or right in the histogram of Figure 6 in order to recalculate the interval boundaries). The following four corrective actions can be taken:

- *Correct left*: $R \leftarrow R + adj$.
- *Correct right*: $R \leftarrow R - adj$.
- *Correct continuously*: If the previous adjustment is “correct left”, then correct left. If the previous adjustment is “correct right”, then correct right.

⁷Monitoring ACF, its changes, and comparing it with the ACF of other processes on-line is very challenging. We are not aware of an efficient way to do that. Currently we opt to compare and monitor its effects, i.e., performance metrics. Monitoring effectively ACF could be a focus of the future work.

- *Correct reversely*: If the previous adjustment is “correct left”, then correct right. If the previous adjustment is “correct right”, then correct left.

The algorithm in Figure 10 describes how the corrective constant R is dynamically adjusted every C requests. Once a new value for R is set, the corrective factors p_i are computed according to the algorithm of Figure 7. Finally, the per server job size boundaries are computed according to Eq. (2) using the recalculated p_i .

4.2.1 Performance of D_EQAL

In this section, we evaluate the effectiveness of D_EQAL. As in the previous sections, each experiment is driven by the WorldCup 10 million request trace, the boundaries of ADAPTLLOAD are computed every $K = 10K$ requests, while the adjustment of the corrective factors for D_EQAL happens every $C = 300K$ requests,

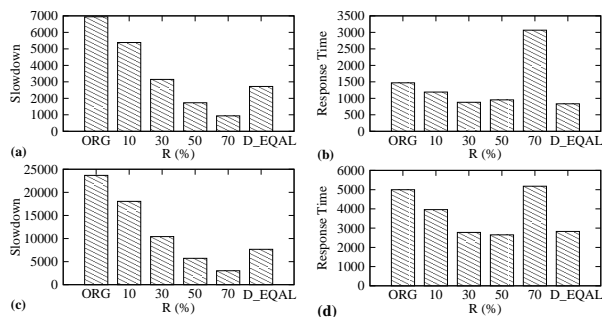


Figure 11. Average slowdown and average response time for the original ADAPTLLOAD, S_EQAL with various values of R , and D_EQAL, under (a)-(b) SRD and (c)-(d) LRD traffic.

We compare the original ADAPTLLOAD, S_EQAL with various values of its corrective constant R , and D_EQAL. Note that in the dynamic policy, we start with a value of $R = 0$, which indicates that we rely on the adaptive algorithm to find the best value of R . Results are presented in Figure 11. Under SRD arrivals, the on-line policy (labeled “D_EQAL”) is comparable to the best performing S_EQAL, where R is set to a set of static values. D_EQAL manages to adjust R such that both slowdown (Figure 11(a)) and response time (see 11(b)) are improved. Similar behavior is observed also under the more challenging LRD traffic. The dynamic policy achieves average slowdown (Figure 11(c)) and response time (see Figure 11(d)) that are several times better than the original ADAPTLLOAD.

Figures 12(a) and (b) show how the value of the corrective constant R changes over time under SRD and LRD arrivals, respectively. Observe how quickly R , that starts from

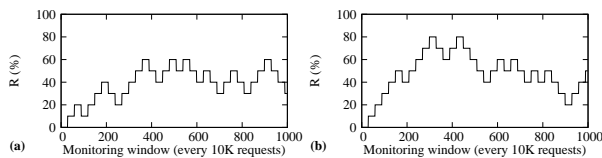


Figure 12. Corrective constant R as a function of time (measured in processed requests) for $C = 300K$ under (a)SRD and (b) LRD traffic.

0, converges toward the best performing for SRD arrivals. For the LRD case convergence is slower. A large value of adj would help find the corrective constant R faster.

5 Conclusions

In this paper, we evaluate the performance of sized-based load balancing policies for homogeneous clustered servers under correlated arrivals. We show that under correlated arrivals sized-based policies, which have been shown to successfully balance load and improve performance when service demands are highly variable, are now ineffective.

Our experiments show that if the arrival process is correlated, then it is not enough for a size-based policy to equally distribute the work among the servers in the cluster – if the arrival streams to individual servers are correlated, then performance significantly degrades. We propose a new size-based load balancing policy, called D_EQAL, that strives to distribute the work such that the load to each server is proportional to the correlation structure of the arrival process to that server and still separates jobs to servers according to their sizes. As a result of this effort, not all servers are equally utilized (i.e., load in the system becomes unbalanced) but this imbalance results in significant performance benefits. D_EQAL does not require any prior knowledge of the correlation structure of the arrival stream or of the job size distribution. Using trace-driven simulation, we show that D_EQAL is an effective on-line policy: by monitoring performance measures it self-adjusts its parameters to transient workload conditions.

References

- [1] D. Abendroth and U. Killat. Intelligent shaping: Well shaped throughout the entire network? In *IEEE INFOCOM 2002*, New York City, NY, June 2002.
- [2] M. Andreolini, M. Colajanni, and R. Morselli. Performance study of dispatching algorithms in multi-tier Web architectures. *Performance Evaluation Review*, 2002.
- [3] M. Arlitt and T. Jin. Workload characterization of the 1998 World Cup Web site. Technical report, Hewlett-Packard Laboratories, Sept. 1999.

- [4] L. Cherkasova, W. Tang, and S. Singhal. An SLA-oriented capacity planning tool for streaming media services. In *Proc. of the International Conference on Dependable Systems and Networks, (DSN-2004)*, Florence, Italy, June 2004.
- [5] K. J. Christensen and V. Ballingam. Reduction of self-similarity by application-level traffic shaping. In *22nd IEEE Conference on Local Computer Networks (LCN '97)*, pages 511–518, Minneapolis, MA, Nov. 1997.
- [6] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Trans. Netw.*, 4(2):209–223, 1996.
- [7] 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, 2005.
- [8] M. E. Gomez and V. Santonja. On the impact of workload burstiness on disk performance. In *Workload characterization of emerging computer applications*, pages 181–201. Kluwer Academic Publishers, 2001.
- [9] M. Harchol-Balter, M. Crovella, and C. Murta. On choosing a task assignment policy for a distributed server system. In *Proceedings of Performance Tools '98, Lecture Notes in Computer Science, Volume 1469*, pages 231–242, Boston, MA, 1998. Springer Verlag.
- [10] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, 15(3):253–285, Aug. 1997.
- [11] L. Kleinrock. *Queueing Systems, Volume I: Theory*. Wiley, 1975.
- [12] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. SIAM, Philadelphia PA, 1999. ASA-SIAM Series on Statistics and Applied Probability.
- [13] N. Mi, Q. Zhang, A. Riska, and E. Smirni. Performance impacts of autocorrelation in tpc-w. Technical Report WM-CS-2005-35, Department of Computer Science, College of William and Mary, Nov. 2005.
- [14] A. Riska and E. Riedel. Analysis of disk-level traces. Technical Report SEA-ARCH-2004-02, Seagate Research, May 2004.
- [15] 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, Nov. 2001.
- [16] U. Vallamsetty, K. Kant, and P. Mohapatra. Characterization of e-commerce traffic. In *WECWIS2002*, Newport Beach, California, 2002.
- [17] F. Xue and S. J. B. Yoo. Self-similar traffic shaping at the edge router in optical packet-switched networks. In *IEEE International Conference on Communications (ICC 2002)*, volume 4, pages 2449–2453, Apr. 2002.
- [18] Q. Zhang, L. Cherkasova, and E. Smirni. Flexsplit: A workload-aware, adaptive load balancing strategy for media clusters. In *Multimedia Computing and Networking (MMCN'06)*, San Jose, CA, Jan. 2006.
- [19] 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, Mar. 2005.