Load Balancing for Performance Differentiation in Dual-Priority Clustered Servers *

Ningfang Mi Qi Zhang
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
{ningfang, qizhang}@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 known 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. We first examine how size-based policies can provide service differentiation and complement admission control and/or priority scheduling policies. We find that under autocorrelated arrivals the effectiveness of size-based policies quickly deteriorates. We propose a two-step resource allocation policy that makes resource assignment decisions based on the following principles. First, instead of equally dispatching the work among all servers in the cluster, the new policy biases load balancing by an effort to reduce performance loss due to autocorrelation in the streams of jobs that are directed to each server. As a second step, an additional, per-class bias guides resource allocation according to different class priorities. As a result, not all servers are equally utilized (i.e., the load in the system becomes unbalanced) but performance benefits are significant and service differentiation is achieved as shown by detailed trace-driven simulations.

1 Introduction

We focus on load balancing in 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.[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. Prior research has shown that the job service time distribution is critical for the performance of load balancing policies in such a setting and that size-based policies, i.e., policies that aim at balancing load based on the size of the incoming jobs, perform well if the goal is to minimize the expected job completion time, job waiting time, and job slowdown [6, 15, 7, 8].

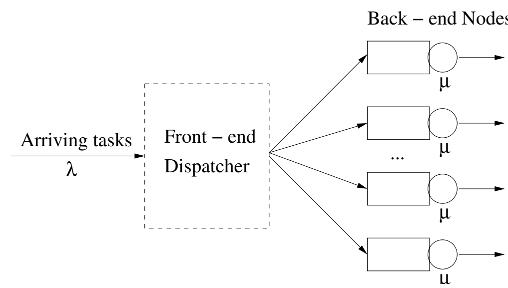

Figure 1. Model of a clustered server.

In this paper, we focus on clustered systems as those depicted in Figure 1 that accept two classes of priority jobs, i.e., high and low priority jobs.[2] Content-distribution networks and media-server clusters that provide streaming of high quality audio and video from a central server configuration are an example of a centralized system where size-based policies provide good balancing solutions [13, 4]. Storage systems which deploy mirroring for enhanced performance and data availability are another case of a clustered system where load balancing based on the job size is beneficial. In both of the above examples, the stream of requests from the system's end-users is considered high priority and served within the delay constraints placed by the respective applications, while the set of system-level activities

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

[1] Throughout this exposition we are using the terms "jobs" and "requests" interchangeably.

[2] In this paper, we focus on a system with two priority classes, for reasons of presentation simplicity. The presented algorithms can be easily extended to several job classes.

that aim at maintaining the cluster and enhancing its performance and availability (via data movement, mirroring, profiling, prefetching) are considered low priority. In such systems, because the streams of requests for the two different priority classes are generated by different processes or applications, their characteristics, i.e., arrivals and service demands, are expected to be different too.

Performance differentiation in such systems can be achieved either via admission control, priority scheduling, or both [5, 3, 10, 2, 16, 9, 12]. The proposed methodologies are often based on feedback control theory, constraint optimization, and preferential scheduling that target at minimizing queuing delays. In this paper, we focus on the problem of performance differentiation in a clustered server from the perspective of load balancing *only*, i.e., we do not consider admission control or priority scheduling to improve on the performance of priority classes. Admission control and priority scheduling, although instrumental for performance differentiation, are outside the scope of this work. Instead, the work presented here can be used as complementary to admission control and priority scheduling, because the results shown can be considered as lower *bounds* to performance, i.e., performance of high priority jobs can only improve if admission control and/or priority scheduling is also deployed.

We focus on a clustered system that accepts two classes of jobs and aim at adjusting size-based load balancing policies to account for performance differentiation. If the arrival stream at the dispatcher of *both* priority classes or *either* of the two classes is *autocorrelated* (i.e., bursty), then the effectiveness of size-based policies deteriorates and policies that "unbalance" the load such that there is a performance bias toward correlated servers become desirable. We further show that when considering performance differentiation, additional per-class load unbalancing that *simply* favors the higher priority class is not sufficient.

Based on our observations, we propose a two-step size-based load balancing policy that aims at reducing the performance degradation due to autocorrelation in each server, while maintaining the property of serving jobs of similar sizes by each server. This new policy, called DIFFEQAL, strives to differentiate services but equally distribute work guided by autocorrelation and load. DIFFEQAL measures autocorrelation of each priority stream in an online fashion and appropriately unbalances load at the cluster aiming at meeting the following two goals: first, the entire load, irrespective of job type, is "shifted" from one server to the next such that the effect of autocorrelation in job performance is minimized and second, per-class load is further "shifted" such that the performance of the high priority class benefits from this shift. This new policy, appropriately unbalances load so that it strikes a balance between two (in some cases) conflicting goals: load is "shifted" such that high priority jobs are moved into less utilized servers, while each server serves requests of as similar size as possible. DIFFEQAL does not assume any *a priori* knowledge of the job service time distribution of the two priority classes, nor any knowledge of the intensity of the dependence structure in their arrival streams. By observing past arrival and service characteristics, the policy adjusts its configuration parameters in an online fashion. To the best of our knowledge this is the first time that load balancing considers both dual-priority jobs *and* dependence in the arrival process as critical characteristics for performance aiming at performance differentiation. The closest work in the literature is the one by Aron et. al. [2] where the problem of load balancing and performance isolation in clustered servers like the one depicted in Figure 1 is addressed by mapping it into an equivalent constrained optimization problem. Our contribution here can be viewed as a mechanism to complement admission control and/or priority scheduling via load balancing.

This paper is organized as follows. Section 2 presents background material and analyzes the performance of size-based policies for dual-priority services. The performance effect of autocorrelation in the arrival streams of the two priority classes for the proposed off-line size-based policies is examined in Section 3. The on-line size-based policy is presented in Section 4. Section 5 summarizes our contributions.

2 Background

In this section we give an overview of the performance effect of autocorrelated traffic in a single queue. We also give a quick overview of EQAL [14], a size-based load balancing policy that have been previously proposed.

2.1 Autocorrelation (ACF)

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 $n = 0, \ldots, \infty$, in discrete time. The ACF, $\rho_X(k)$, and the CV are defined as follows

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

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

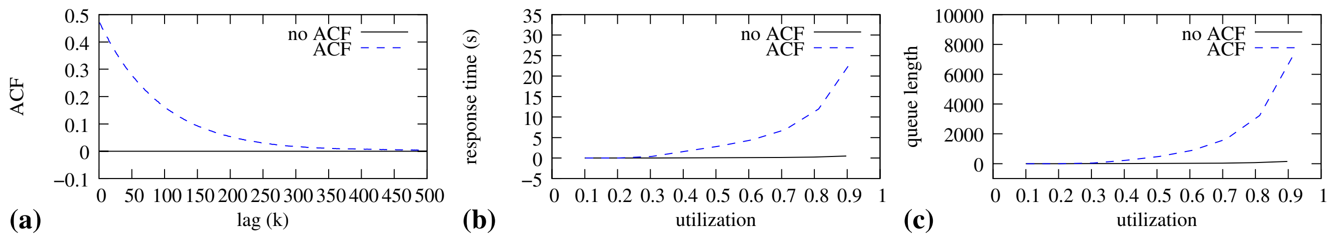

Figure 2. ACF of the inter-arrivals (a), response time (b) and queue length (c) as a function of system utilization when inter-arrivals are independent (no ACF) or have positive autocorrelation (ACF).

most cases ACF approaches zero as k increases. CV values less than 1 indicate that the variability of the sample is low and CV values larger than 1 show high variability.

Autocorrelated arrivals are observed in different levels of real systems, such as the incoming traffic to e-commerce Web servers [1], or the arrivals at storage systems supporting (dedicatedly) various applications [14]. Using the data from the storage system of a Web server described in [14], we parameterize a simple MMPP/H_2/1 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.[3] The service process is drawn from a 2-stage hyper-exponential (H_2) distribution with mean service time equal to 3 ms and CV equal to 1.85. 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), or there is positive autocorrelation with ACF starting at 0.47 at lag=1 but decaying to 0 at lag=500 (see Figure 2(a)).

Figure 2 presents performance measures for the MMPP/H_2/1 queuing model as a function of system utilization. We measure performance by reporting on response time (see Figure 2(b)) which is the sum of the request service time and its waiting time in the queue, and queue length (see Figure 2(c)) which is the total number of requests in the server queue including the one in service. Observe that system performance deteriorates by 3 orders of magnitude when comparing to the case with no ACF arrivals.[4] Hence, it is not only variability in the arrival and service processes that hurts performance, but more importantly the dependence structure in the arrival process.

[3]We selected a Markovian-Modulated Poisson Process (MMPP), a special case of the Markovian Arrival Process (MAP) [11], to model autocorrelated inter-arrival times because it is analytically tractable.

[4]Because of the scale used in the figure and because of the difference of the two curves, the performance measures with no ACF look flat. With no ACF for utilization equal to 0.9, queue length is equal to 152, but this number is dwarfed in comparison to the queue length with autocorrelated arrivals.

2.2 EQAL

In prior work, a size-based policy EQAL that does not require *a priori* knowledge of the service time distribution has been shown to be effective under correlated traffic conditions [14]. EQAL accounts for dependence in the arrival process by relaxing traditional load balancing goal to balance the work among all nodes of the cluster and has demonstrated superior performance under correlated traffic [14]. The policy is summarized as follows:

- **EQAL:** In a cluster with N server nodes, EQAL partitions the possible request sizes into N intervals, $\{[s_0 \equiv 0, s_1), [s_1, s_2), \ldots [s_{N-1}, s_N \equiv \infty)\}$, so that if the size of a requested file falls in the ith interval, i.e., $[s_{i-1}, s_i)$, this request is routed to server i, for $1 \le i \le N$. These boundaries s_i for $1 \le i \le N$ are determined by constructing the histogram of request sizes and by weighting the work assigned to each server with the degree of autocorrelation in the arrival process, which is based on the observation that in order to achieve similar performance levels under autocorrelated arrivals the system utilization must be lower than the utilization under independent arrivals. A shifting percentage vector $\mathbf{p} = (p_1, p_2, \cdots, p_N)$ is defined in EQAL so that the work assigned at server i is equal to $(1 + p_i)\frac{\bar{S}}{N}$ for $1 \le i \le N$, provided that $\sum_{i=1}^{N} p_i = 0$ for $1 \le i \le N$. Here $\bar{S}$ is the amount of total work arriving the cluster. The values of p_i for $1 \le i \le N$ are statically defined by letting p_1 be equal to a pre-determined corrective constant R, $0 \le R < 100\%$. The rest of the shifting percentages p_i, for $2 \le i \le N$, are calculated using a semi-geometric increasing method [14]. The following equation formalizes this new load distribution:

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

where $F(x)$ is the CDF of the request sizes. Note that when $R = 0$, EQAL is a special size-based load balancing policy ADAPTLOAD [15], which aims at partitioning the total work equally. For a transient workload, the value of the $N-1$ size boundaries $s_1, s_2, \ldots, s_{N-1}$ is critical. EQAL self-adjusts these boundaries by predicting the incoming workload based on an exponentially discounted history of the last K requests.[5] In our simulations, we set the value of K equal to 10000.

We evaluate EQAL under the correlated traffic by analyzing the response time (i.e., wait time plus service time), and the request slowdown (i.e., the ratio of the actual response time of a request to its service time). 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.[6] The inter-arrivals of the low priority class have an ACF structure which is the same as the one in Figure 2(a), while the high priority class has independent arrival process. Both arrival processes have the same CV of 4.47. Both service processes are independent and have the same mean, but different CVs that are set to 1.87 and 10 for low priority and high priority classes, respectively.

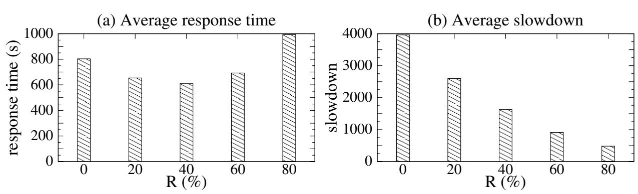

Figure 3. The average response time (a) and the average slowdown (b) of EQAL by different R under correlated arrivals.

Figure 3 shows system performance under this correlated traffic in the cluster. Both average slowdown and average response time reduce as the shifting ratio increases, but a turning point exists where shifting more work to subsequent servers adversely affects average response time. The best performance is achieved when the shifting ratio of EQAL is 40%.

Despite of its good performance under correlated arrivals, EQAL treats all requests equally, i.e., without distinguishing job priorities. In the following section, we present a two-step load balancing policy that provides service differentiation for the two classes of jobs.

[5] For more details on the way the policy self-adjusts its boundaries to changing workload conditions, we direct the readers to [14].

[6] Experiments 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.

3 Two-step Resource Allocation Policy

In this section, we propose an enhancement to EQAL, to account for dependence in the arrival process and provide service differentiation by relaxing their basic goal to balance 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 the nodes with weaker correlation in their inter-arrival times, and unfairly *shift* per-class loads such that high priority jobs are moved into less utilized servers. In the following sections, we first present an off-line version of the policy where we assume a priori knowledge of the dependence structure in the arrival streams. Then, we present an on-line version of this policy where past arrival and service characteristics guide the adjustment of configuration parameters to improve overall system performance.

3.1 Off-line DIFFEQAL

Recall that with appropriate shifting parameters, EQAL gives the optimal overall performance for both average response time and average slowdown. However, EQAL does not provide performance differentiation because it only uses one histogram for both classes of jobs.

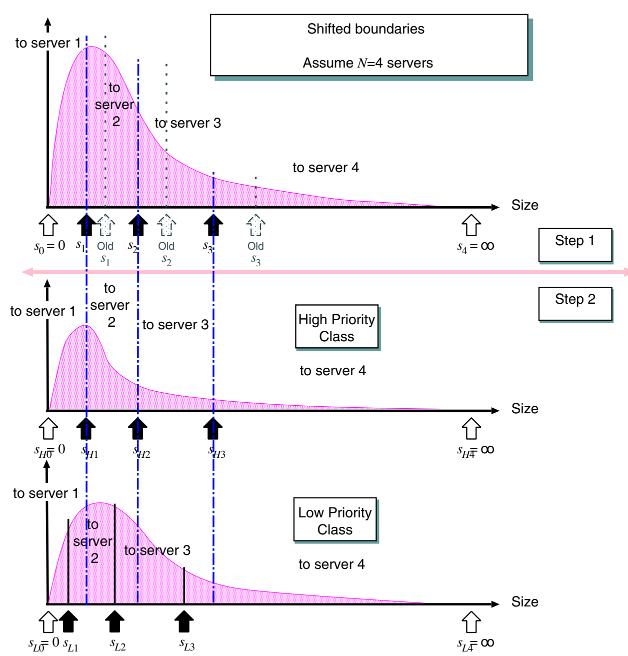

Figure 4. DIFFEQAL's high level idea for recalculating boundaries under autocorrelated inter-arrival times and different priority classes.

Step 1.1		initialize variables	
	a.	initialize a variable $adjust$	$adjust \leftarrow -R$
	b.	initialize the shifting percentages	$p_i \leftarrow 0$ for all $1 \leq i \leq N$
Step 1.2		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	$adjust \leftarrow adjust/2$
Step 2.1		initialize variables for per class	
	a.	initialize a variable $adjust^c$	$adjust^c \leftarrow -R^c$ for $c \in \{high, low\}$
	b.	initialize the shifting percentages	$p_i^c \leftarrow 0$ for all $1 \leq i \leq N$
Step 2.2		for $i = 1$ to $N-1$ do	
	a.	add $adjust^c$ to p_i^c	$p_i^c \leftarrow p_i^c + adjust^c$
	b.	for $j = i+1$ to N do	
		equally distribute $adjust^c$ to the remaining servers	$p_j^c \leftarrow p_j^c - \frac{adjust^c}{N-i}$
	c.	reduce $adjust^c$ to half	$adjust^c \leftarrow adjust^c/2$

Figure 5. The algorithm for setting the shifting percentages p_i and p_i^c for dual-priority classes in DIFFEQAL.

Off-line DIFFEQAL consists of two steps, as depicted in Figure 4. The first step of DIFFEQAL is equivalent to EQAL, i.e., it moves (both high and low priorities) jobs from the servers with a strongly correlated arrivals to the servers with weakly correlated arrivals. As a second step, an additional, per-class bias guides load balancing according to different class priorities. We introduce a per-class corrective factor vector $\mathbf{p}^c$, where $c \in \{high, low\}$, so that we have the following equation for the work of class c assigned at server i:

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

where $F^c(x)$ is the CDF of the request sizes of class c and $\bar{S}_i^c$ is the amount of the work belonging to class c, which is assigned to server i after the first step. Note that p_i^c can take both negative and positive values and that equation $\sum_{i=1}^{N} p_i^c = 0$ should be satisfied for each class.

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, where $0 \leq R < 100\%$, and then by calculating the rest of the corrective factors p_i for $2 \leq i \leq N$ using a semi-geometric increasing method, as described by the algorithm in Figure 5, Step 1. Note that R equal to 0 performs best under independent traffic, while $R > 0$ outperforms under correlated traffic [14]. Because most requests are for small files and the first server receives most of requests, the ACF of its arrival process is very similar to the original ACF of the arrival process at the dispatcher. It follows that the corrective parameter p_1 is usually negative, i.e., $p_1 = -R$. For example, if we define $R = 10\%$ then the corrective parameters for a 4-server cluster are $p_1 = -10\%$, $p_2 = -1.67\%$, $p_3 = 3.33\%$ and $p_4 = 8.34\%$.

In order to favor high-priority jobs while improving overall system performance, we continue to determine the values of the per class corrective factors p_i^c ($c \in \{high, low\}$), by letting p_1^c be equal to a pre-determined corrective constant R^c, where $0 \leq R^c < 100\%$, and then by calculating the rest of the corrective parameters p_i^c for $2 \leq i \leq N$, using the same semi-geometric increasing method as for computing p_i (see the algorithm in Figure 5, Step 2). Note that adjusting both R^{low} and R^{high} concurrently makes system performance less predictable. Consequently, we fix one class boundaries and control the performance differentiation by shifting the other one only. For example, we fix the parameters of the high priority class, resulting in $R^{high} = 0$ and $p_i^{high} = 0$, for $1 \leq i \leq N$. Because the first server is usually the one that exhibits strongly correlated arrivals, the corrective parameter p_1^{low} is negative, i.e., $p_1^{low} = -R^{low}$, to ensure that most high-priority small jobs are served at servers with lower utilization.

3.2 Performance Evaluation of the off-line DIFFEQAL

We evaluate DIFFEQAL using arrivals of two classes, where potentially each class has different inter-arrival/service time distributions. The policy effectiveness is examined by using both independent and correlated arrivals. In all our experiments, the entire system utilization is 50%.[7] The total sample space is 10 million requests.

I. No ACF in the arrivals of both classes

The first set of experiments examines independent arrivals. In all experiments, the autocorrelated structure and CV of inter-arrival times for both classes are the same, i.e., ACF = 0 for all lags and CV = 4.47, but the mean arrival rates are different, which results in a per-class load ratio of dual classes equal to 70% (low priority) over 30% (high priority).[8] The mean service times of these two classes are also the same, but we use different CVs to illustrate the effect of service variability on system performance.

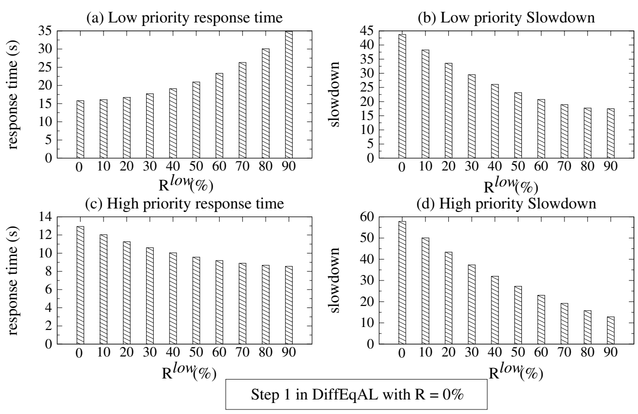

Figure 6. Average per-class response time and request slowdown of DIFFEQAL by different R^{low} under independent arrivals. R in the first step is set as 0. The low priority class has CV equal to 1.87 and the high priority class has CV equal to 10.

Figure 6 gives the performance results when the low priority class has a CV of only 1.87, but the high priority class requests are highly variable with a CV equal to 10.[9] Under this setting, the best overall performance following the first step in DIFFEQAL, is for $R = 0$, which is effectively EQAL with $R = 0$ [14]. Therefore, in this set of experiments, we set corrective constant R in Figure 5 Step 1 as zero. Without considering performance differentiation, dual classes have similar performance results. We then further shift the low priority class jobs to the latter servers as described in the second step of DIFFEQAL. Note that for $R^{low} = 0$, DIFFEQAL becomes identical to EQAL. By increasing the shifting percentage R^{low}, the average response time and the average slowdown of the high priority class keep decreasing (see Figure 6(c)-(d)). For instance, when $R^{low} = 90\%$, the values of the average response time and the average slowdown are equal to 8.5 and 12.8, respectively, which are about 66% and 22% of those under EQAL with $R = 0$. This improvement however negatively affects the performance of the low priority class, whose average response time increases by 2 times (see Figure 6(a)), but its average slowdown improves by 60% (see Figure 6(b)) as a result of DIFFEQAL's shifting. Note that small jobs, which have large chance for huge slowdown values, are still served in the first server. On the other hand, the incremental response time of comparatively fewer large jobs served in the last server may only increase their slowdown slightly.

Experiments use the same ACF in the inter-arrival streams of both jobs were also done and the performance differentiation trends are qualitatively the same as those experiments with independent arrivals.

Observation 1 *If both priority classes have the same arrival process with or without autocorrelation, then shifting the size boundaries of the low priority class improves average response time and average slowdown of the high priority class. Such improvement is more significant and more effective when high priority jobs have highly variable sizes.* [10]

II. Low priority class has correlated arrival process; high priority class has independent arrival process

We now consider the correlated experiments, where the inter-arrivals of the low priority class have an autocorrelated structure as the one in Figure 2(a), the arrivals for high priority class are independent, and the high priority requests have higher variable service times. Under this setting, EQAL gives the best performance when $R = 40\%$ (see Figure 3).

We thus set R value in the first step of DIFFEQAL equal to 40% when both priority classes or either of the two classes is autocorrelated. Figure 7 illustrates the performance differentiation achieved by DIFFEQAL as a function of different R^{low} values. Note DIFFEQAL with $R^{low} = 0$ and $R = 40\%$ is effectively EQAL with $R = 40\%$. Due to its correlated inter-arrivals, the low priority class has worse performance even without per-class shifting. Both of its average response time and average slowdown are 2 times higher than the high priority performance. As R^{low} increases, the average response time of the high priority class keeps constant till $R^{low} = 40\%$, but its average slowdown keeps decreasing to 36%. When $R^{low} = 60\%$, average response time increases by 15%, but average slowdown reaches its ideal value with 77% improvement.

[7]Experiments under light and heavy loads are also evaluated and provide qualitatively similar results as that under medium load.

[8]Throughout this paper, we use this load ratio for all the experiments. Other ratios give qualitatively the same results so that we do not report them here due to lack of space.

[9]In all the experiments, we use a CV equal to 10 as a high variance, and a CV equal to 1.87 as a low variance.

[10]Experiments with low priority class having CV equal to 10 and the high priority class having CV equal to 1.87 have also been done but the improvement is less effective and we do not reported them due to lack of space.

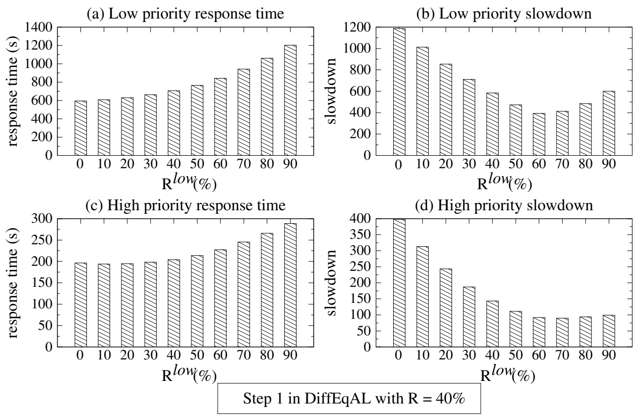

Figure 7. Average per-class response time and request slowdown of DIFFEQAL by different R^{low} under correlated low priority arrivals. R in the first step is set as 40%. The low priority class has CV equal to 1.87 and the high priority class has CV equal to 10.

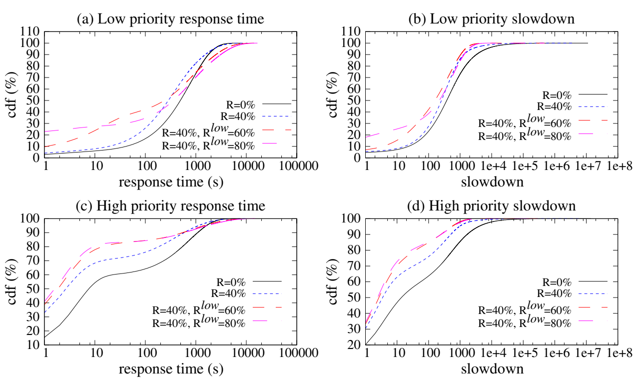

Figure 8. The CDFs of per-class response time and request slowdown of EQAL for different R and DIFFEQAL for different R^{low} under correlated low priority arrivals. The low priority class has CV equal to 1.87 and the high priority class has CV equal to 10.

We then look into the cumulative probability function (CDF) of per-class results to better understand the per class policy behavior. Figure 8 gives the CDFs of response time and slowdown for both classes. The higher the line, the better the policy performs. Across all graphs in Figure 8, the EQAL for $R = 0$ performs worst. DIFFEQAL with various corrective constants R^{low} provides better slowdown for the high priority jobs than EQAL with $R = 40\%$ does (see Figure 8(d)). Additionally, DIFFEQAL also provides better slowdown for the low priority class except for $R^{low} = 80\%$ (see Figure 8(b)). Although $R^{low} = 60\%$ does not give the best average response time for the high priority class, it improves the response time of most requests as shown in Figure 8(c). Compared with no per-class shifting,(i.e., the line labeled "$R = 40\%$"), 12% more of the total high priority requests have response time less than 50 under $R^{low} = 60\%$. Its higher average response time can be explained by its long tail of the CDF of response times, but admission control or priority scheduling can further improve on the tail performance.

III. High priority class has correlated arrival process; low priority class has independent arrival process

This set of experiments considers the cases where ACF exists in the high priority class. Other parameters are kept the same as in the previous experiment. Again the ideal EQAL is for $R = 40\%$. We thus set R value in the first step of DIFFEQAL equal to 40%. The results are displayed in Figure 9. The best high priority performance is achieved under the most aggressive shifting, $R^{low} = 90\%$. Note that after the high priority class is favored by shifting low priority jobs to the latter servers, the high priority class still performs worse than the low priority class even under the best R^{low}, showing that shifting only is not sufficient to maintain the acceptable performance. In this case, admission control may be the only way to improve performance. Indeed, experiments that dropped *all* low priority requests show that performance improvements are still incremental. It is the ACF structure of the high priority class that causes performance degradation.

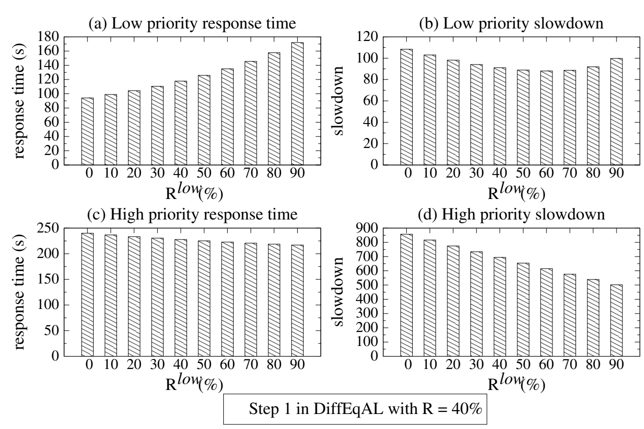

Figure 9. Average per-class response time and request slowdown of DIFFEQAL by different R^{low} under correlated high priority arrivals. R in the first step is set as 40%. The low priority class has CV equal to 1.87 and the high priority class has CV equal to 10.

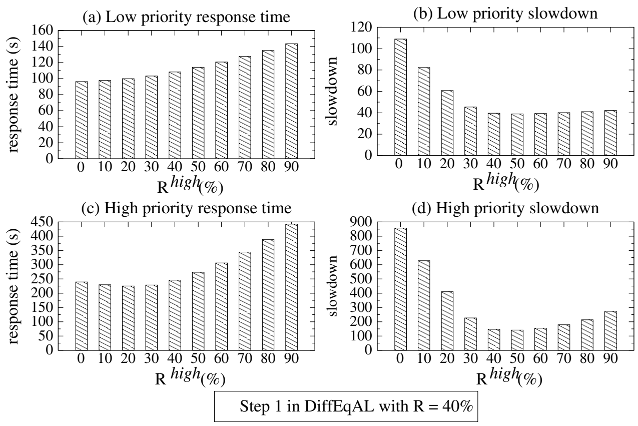

Figure 10. Average per-class response time and request slowdown of DIFFEQAL by different R^{high} under correlated high priority arrivals. R in the first step is set as 40%. The low priority class has CV equal to 1.87 and the high priority class has CV equal to 10.

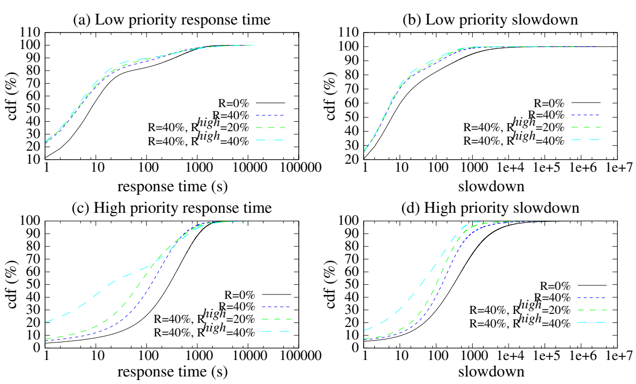

Figure 11. The CDFs of per-class response time and request slowdown of EQAL with different R and DIFFEQAL for different R^{high} under correlated high priority arrivals. The low priority class has CV equal to 1.87 and the high priority class has CV equal to 10.

By focusing on the effect of autocorrelation structure, we now opt to shift the class with more autocorrelated arrivals, i.e., the high priority class. As shown in Figure 10(c), the average response time of the high priority class is kept stable till $R^{high} = 40\%$, and then it increases quickly, as confirmed also by the CDF results shown in Figure 11. When $R^{high} = 20\%$, 24.7% of the high priority requests have response times less than 20, while for $R^{high} = 40\%$, this percentage increases to 51%.

Comparing Figures 9 and 10, we conclude that shifting the high priority class gives better performance when stronger ACF exists in the high priority class. Also note that although the two classes still have different variation in their service times, this does not affect which class to shift.

Observation 2 *When the two classes have different ACF structures, always shifting the one with a stronger ACF yields better performance.*

4 On-line service differentiation via load balancing

In the previous section, we confirmed that by further unbalancing load of low-priority jobs in a system that deploys a size-aware load balancing policy, the performance of the high-priority class improves. However, if the autocorrelation structure of arrivals of the high priority class is stronger than the autocorrelation in arrivals of the low priority class, then unbalancing the load of the high-priority class is more beneficial *both* for overall and per-class performance than simply unbalancing the load of the low-priority class only. Consequently, autocorrelation is identified as more important for performance than service time variation. When the ACF structure of the arrivals of the two classes of jobs is substantially different, identifying which class should be unbalanced for better performance becomes critical. Here, we propose a new on-line version of DIFFEQAL which does not assume any *a priori* knowledge of the workload characteristics. Our prediction is based on monitoring past arrival and service processes. By observing past arrival and service characteristics, the policy measures the autocorrelation of each priority stream and then adjusts its configuration parameters, e.g., corrective factors for both classes, in an online fashion.

The policy updates its parameters for every C jobs served by the cluster. C must be large enough to allow for effective ACF measurement but also small enough to allow for quick adaptation to transient workload conditions. In the experiments presented here C is set to 100K. The policy starts by setting corrective constants R, R^{high} and R^{low}, to zero, i.e., the load is equally distributed into each server of the cluster. After every C jobs, the policy computes the ACF of each priority class using the observed inter-arrival times of jobs within the batch. The measured ACF is used as prediction for batch of the next C jobs. Based on the predicted ACF per priority class, the policy resets the corrective constants R, R^{high} and R^{low} to the appropriate predetermined values $shift$, $shift^{high}$, and $shift^{low}$, respectively. In our experiments we set $shift$, $shift^{high}$, and $shift^{low}$ equal to 40%, 20% and 40%, respectively. The following four scenarios of ACF in the arrivals of the priority classes are considered:

1. *neither* priority class is autocorrelated:
- $R \leftarrow 0$
- $R^{high} \leftarrow 0$, $R^{low} \leftarrow shift^{low}$

2. two priority classes have *similar* ACF:
- $R \leftarrow shift$
- $R^{high} \leftarrow 0$, $R^{low} \leftarrow shift^{low}$

3. *low* priority class has stronger ACF:
- $R \leftarrow shift$
- $R^{high} \leftarrow 0$, $R^{low} \leftarrow shift^{low}$

4. *high* priority class has stronger ACF:
- $R \leftarrow shift$
- $R^{low} \leftarrow 0$, $R^{high} \leftarrow shift^{high}$

Corrective factors p_i^c, where $1 \leq i \leq N$ and $c \in \{high, low\}$, are computed using the algorithm of Figure 5. Once all the corrective factors are computed, the per server and per class job size boundaries are calculated using Eq. (1) and (2). The online part of the load balancing algorithm is described in Figure 12.

1 initialize
  a. set $R \leftarrow 0$
  b. set $R^c \leftarrow 0$ for $c \in \{high, low\}$
2 every C requests
  a. compute the ACF of each priority class
  b. **if** *neither* priority class has ACF
    then $R \leftarrow 0$
    else $R \leftarrow shift$
  c. **if** *high* priority class has stronger ACF
    then **I.** $R^{low} \leftarrow 0$
      II. $R^{high} \leftarrow shift^{high}$
    else **I.** $R^{high} \leftarrow 0$
      II. $R^{low} \leftarrow shift^{low}$
3. compute p_i and p_i^c for $1 \leq i \leq N$ using Figure 5
4. every K requests
  compute per server per class job size boundaries using Eq. (1), Eq. (2) and the p_i, p_i^c computed in **3.**
5. goto **2.**

Figure 12. Reseting of the corrective constants R, R^{high} and R^{low} in on-line fashion.

4.1 Performance of On-line DIFFEQAL

In this section, we evaluate the effectiveness of on-line DIFFEQAL. As in the previous sections, each experiment is driven by the 10 million request trace consisting of 7 million low priority requests and 3 million high priority requests. The CV of the service time of low priority class is set to 1.87 and the CV of the service time of the high priority class is equal to 10, the boundaries are computed every 10K requests, and the resetting of corrective constants R, R^{high} and R^{low} for on-line DIFFEQAL is triggered every $C = 100K$ requests. Additionally, in this trace, the autocorrelation of each class stream alternates as follows: in the first 2 million requests only the low priority class is autocorrelated, then in the next 2 million requests only the high priority class is autocorrelated. The on-line DIFFEQAL policy alternates $R = shift$, $R^{low} = shift^{low}$ and $R^{high} = 0$ with $R = shift$, $R^{low} = 0$ and $R^{high} = shift^{high}$.

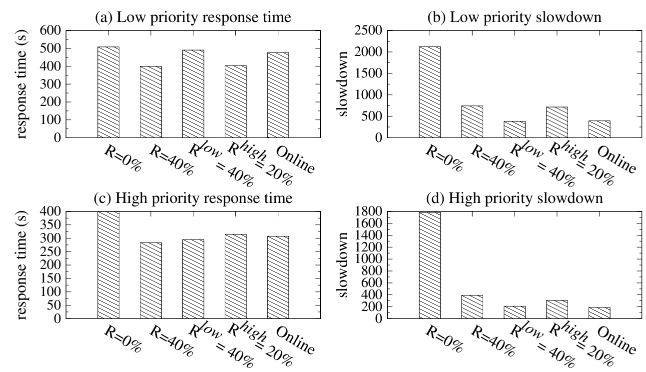

Figure 13. Average per-class response time and request slowdown for EQAL with $R = 0$, EQAL with $R = 40\%$, off-line DIFFEQAL with $R = 40\%, R^{high} = 0$ and $R^{low} = 40\%$, off-line DIFFEQAL with $R = 40\%, R^{low} = 0$ and $R^{high} = 20\%$, and on-line DIFFEQAL under mixed autocorrelated traffic.

We compare the overall performance of on-line DIFFEQAL with those of EQAL for $R = 0$ and $R = 40\%$, off-line DIFFEQAL with $R^{low} = 40\%$, and off-line DIFFEQAL with $R^{high} = 20\%$. Note that R is equal to 40% for all DIFFEQAL experiments. The results are presented in Figure 13. Consistent to the performance results shown in the previous sections, the effectiveness of EQAL with $R = 0$, i.e., equally load balancing policy, quickly deteriorates under correlated traffic while EQAL with $R = 40\%$ achieves significant performance improvement. Although EQAL for $R = 40\%$ achieves the fastest average response time for both classes, off-line DIFFEQAL achieves the smallest average request slowdown for both classes. The on-line DIFFEQAL balances the average response time and the average request slowdown, i.e., both are close to the optimal results.

In Figure 14, the CDFs of per-class response time and request slowdown are shown. These CDFs further confirm that the on-line DIFFEQAL achieves better per-class performance than EQAL for most requests, especially for small requests. Using on-line DIFFEQAL, about 65% of high-priority requests have response time less than 50 and about 50% of low-priority requests have less response time than 50. Most importantly, the on-line DIFFEQAL policy achieves the best performance differentiation, with a clear

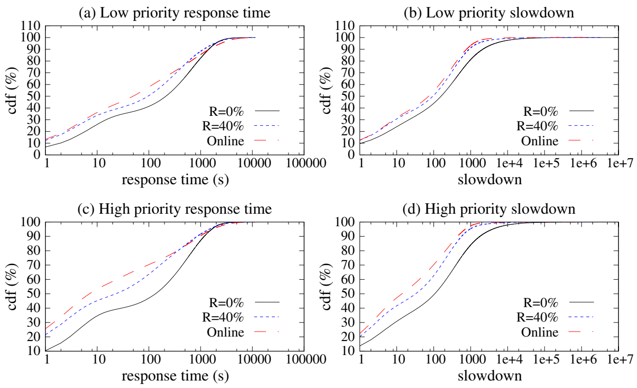

Figure 14. The CDFs of per-class response time and request slowdown for EQAL with $R = 0$, EQAL with $R = 40\%$, and on-line DIFFEQAL under mixed autocorrelated traffic.

performance bias toward the high-priority class.

5 Conclusion

We presented a size-aware load balancing policy that, in addition to distributing the load among servers of a cluster, differentiates service for different priority classes. The new policy, call DIFFEQAL, incorporates into its decision making salient workload characteristics, such as the ACF in arrivals and the variability in service demands, as well as the workload user- or system-defined priority. While DIFFEQAL allocates cluster resources aiming at service differentiation between different priority classes, it can be used as complementary to admission control or priority scheduling mechanisms in the system.

DIFFEQAL aims at meeting two conflicting goals: unbalance work across servers under correlated arrivals while aiming at reducing the per-server demand variability and distinguish the different priority classes in the cluster workload, i.e., improve on high priority class performance but maintain low-priority class performance. DIFFEQAL differentiates service by further unbalancing the load of the classes that exhibit correlated arrivals.

We also present an on-line version of DIFFEQAL, which monitors the workload and successfully predicts the correlation structure of future arrivals, and finally adjusts its parameters based on these predictions. Our simulation evaluation indicates that under highly changing workloads the on-line DIFFEQAL adapts its parameters well to incoming workload and performs nearly as a static policy with a priori workload knowledge. Our future work is to improve the adaptiveness of DIFFEQAL aiming at self-adjusting its parameters to transient workload conditions by monitoring performance measures.

References

[1] M. Arlitt and T. Jin. Workload characterization of the 1998 World Cup Web site. Technical report, Hewlett-Packard Laboratories, Sept. 1999.

[2] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: a mechanism for resource management in cluster-based network servers. In *SIGMETRICS*, pages 90–101, 2000.

[3] N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, 13(5):64–71, 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] L. Eggert and J. S. Heidemann. Application-level differentiated services for web servers. *World Wide Web*, 2(3):133–142, 1999.

[6] 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.

[7] M. Harchol-Balter, M. Crovella, and C. Murta. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59(2):204–228, Nov. 1999.

[8] M. Harchol-Balter, B. .Schroeder, N. Bansal, and M. Agrawal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems*, 21(2):207–233, May 2003.

[9] Y. Huang and R. Guérin. A simple fifo-based scheme for differentiated loss guarantees. In *International Workshop on Quality of Service (IWQoS2004)*, pages 96–105, 2004.

[10] V. Kanodia and E. Knightly. Multi-class latency-bounded web services. In *International Workshop on Quality of Service (IWQoS2000)*, Pittsburgh, PA, June 2000.

[11] 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.

[12] V. Liberatore. Local flow separation. In *International Workshop on Quality of Service (IWQoS2004)*, pages 87–95, 2004.

[13] 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.

[14] Q. Zhang, N. Mi, A. Riska, and E. Smirni. Load unbalancing to improve performance under autocorrelated traffic. In *Proceedings of the 26th International Conference on Distributed Computing Systems (ICDCS2006)*, Lisboa, Portugal, July 2006.

[15] 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.

[16] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation in cluster-based network servers. In *INFOCOM*, pages 679–688, 2001.