Autocorrelation-Driven Load Control in Distributed Systems

Ningfang Mi
Northeastern University
Boston, MA, USA

Giuliano Casale
SAP Research
CEC Belfast, UK

Qi Zhang
Microsoft
Seattle, WA, USA

Alma Riska
Seagate Research
Pittsburgh, PA, USA

Evgenia Smirni
College of William and Mary
Williamsburg, VA, USA

***Abstract*—In this paper, we propose a new approach for the development of load control policies in autonomic multi-tier systems. We control system load in a completely new way compared to existing policies: we leverage on the autocorrelation of service times and show that autocorrelation can be used to forecast future service requirements of requests and adaptively control system load. To the best of our knowledge, this is the first direct application of autocorrelation of service times to autonomic load control.**

We propose ALOC and D_ALOC, two autocorrelation-driven policies that drop a percentage of the load in order to meet pre-defined quality-of-service levels in a distributed system. Both policies are easy to implement and rely on minimal assumptions. In particular, D_ALOC is a fully no-knowledge measurement-based policy that self-adjusts its load control parameters based only on policy targets and on statistical information of requests served in the past. We illustrate the effectiveness of these new policies in a distributed multi-server setting via detailed trace driven simulations. We show that if these policies are employed in the server with a temporal dependent service process, then end-to-end response time, across all servers, reduces up to 80% by only dropping at most 13% of the incoming requests. Using real traces, we also show that, in the constrained case of being able to drop only from a portion of the incoming workload, our policy still improves request response time by up to 30%.

Keywords: autocorrelated flows, autonomic systems, self-adaptive policies, load control, distributed systems

I. Introduction

Recent work in Web systems, such as Internet servers [21], multi-tier architectures [4], and online Data Stream Management Systems (DSMSs) [7], has drawn attention to the problem of defining effective load control techniques to keep a system responsive under a variety of overload conditions, e.g., Internet flash crowds, large data arrival streams triggered by environmental sensors. We here focus on the fundamental issue of performing load control when the system *cannot* proactively inspect the incoming workload and decide which requests are the most important or cost-effective to serve. For example, a Web server that operates in a virtualized environment is often forced to shed work arbitrarily because it is unable to preventively inspect the performance effects of its predicted workload [21]. Therefore, it is important to define effective low-overhead strategies that autonomically control the incoming load based on the server past history. In this paper, we study the problem of load control under the assumption that there is no knowledge of the incoming workload characteristics. We examine the feasibility of driving the system load control by the temporal dependent characteristics of the past processed requests, which can be determined with minimal computational effort. In particular, we focus on servers accepting requests from an open environment, but having locally a limited buffer size to host requests waiting to be served; these systems, which include among other multi-tier Web systems, are commonly modeled as closed queuing systems [11], [20], [12].

In computer systems, temporal dependence in service of one of the queues is a characteristic that significantly degrades performance in addition to well-known facts like workload variability [3], [13]. For example, in systems that operate using the standard multi-tiered paradigm, temporal dependence has been located in the service process of the front server [13] or of the back-end database [12] and is an effect of the hardware/software configuration of the system. Such temporal dependence in one server of the entire system results in dramatic end-to-end degradation in response times [13].

In this paper, we show that temporal dependence can be exploited to forecast future service requirements of requests and we use this prediction as an additional criterion for resource allocation. The structure of temporal dependence in service can be described and quantified via the *autocorrelation function* (ACF) [2]. The ACF essentially captures the "ordering" of random values in a time series. High positive ACF values imply strong temporal locality: a value of the random variable in the time series has a high probability to be followed by another variable of the same order of magnitude, while negative ACF values imply the opposite. Intuitively, ACF can be used to quantify burstiness in service times and to forecast service requirements in the near future [2].

Autocorrelated flows have been identified as an important traffic characteristic in communication networks [15] and have fueled much research over the past two decades. From a queueing perspective, it has been shown that dependence in the arrival process of a single queue results in dramatic performance degradation [6], [1]. The presence of autocorrelation has been only recently identified as prevalent in a variety of settings [16], [13], [9], [18]. The analysis of systems in presence of autocorrelated service is particularly challenging: autocorrelation in the service process of one of the queues affects request burstiness throughout the system and significantly degrades end-to-end system performance [13].

In this paper, we propose to use autocorrelation to selec-

tively drop requests that are most harmful to performance. We design a new policy, called ALOC, for the autocorrelation-driven load control in autonomic systems. The static version of this policy assumes no knowledge of the length of queued jobs, but requires a priori knowledge of the autocorrelation function of the service process. ALOC is triggered when the queue length grows beyond a certain threshold *and* the last served job was long. Using an autocorrelation-based heuristic, ALOC drops some jobs from the queue to improve the round-trip times of other requests. Detailed simulations using synthetically generated workloads illustrate the effectiveness of this heuristic policy for load control: dramatic reductions to end-to-end average response times up to 80% with a drop rate upto 13% only.

We also propose a dynamic version of the ALOC policy called D_ALOC that is truly a *no knowledge* policy, i.e., it does not assume any a priori knowledge of the autocorrelation function or of the queue length threshold that triggers selective job drop. This policy successfully measures autocorrelation in an online fashion and self-adjusts its triggering parameter according to the measured performance and autocorrelation strengths. The effectiveness and robustness of D_ALOC is shown via detailed experimentation using synthetically generated workloads as well as actual traces measured at the disk level of a streaming system.

The stated goals and outline of this work can therefore be summarized as follows:

- to discuss the performance effects due to an autocorrelated service process in systems, which motivates the development of the ALOC policy (Section II),
- to develop the ALOC and D_ALOC policies which reduce load on an autocorrelated queue based on online monitoring of the system and evaluate its performance via simulation with synthetic traces (Section III),
- to validate the effectiveness and robustness of the new proposed policies via trace-driven simulations using real traces (Section IV).

In Section V, we summarize our findings and outline future work.

II. Motivating Example

The performance of many real systems, such as multi-tiered systems, can be accurately described in terms of closed queueing systems [11]. Closed models have been intensively studied in performance evaluation, but they are usually very hard to evaluate or optimize when the workloads have time-varying characteristics. In the next sections, we focus on the latter issue of optimizing the performance of real systems where service times exhibit *temporal dependence*. Under temporal dependence, consecutive service times cannot be assumed independent of each other (e.g., a long service time is more likely to be followed by another long service time) resulting in burstiness which consistently degrades performance. We propose in this section a motivating example to show the negative effects of temporal dependence in systems.

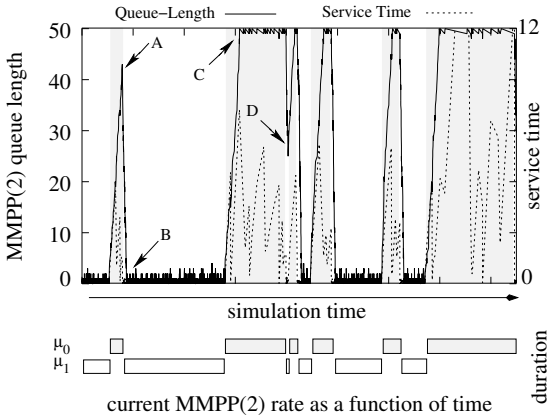

Fig. 1. Queue-length and service times measured in the simulation case study. In shaded areas, jobs are completed with slow rate μ_0. Samples of the duration A_k for $k = \{0, 1\}$ are shown below the main diagram.

We investigate how temporal dependence affects system performance in a closed system. Throughout the example, we quantify temporal dependence by the autocorrelation function (ACF) of service times. Let $\{X_t\}$ be a stationary time series of identically distributed random variables, where $t = 0, 1, 2, \ldots, \infty$. The autocorrelation function ρ_j is the sequence of correlation coefficients:

$$\rho_j = E[(X_{t-j} - \mu^{-1})(X_t - \mu^{-1})]/\sigma^2, \qquad (1)$$

where μ^{-1} is the mean and σ^2 is the variance of $\{X_t\}$; the subscript j is called the lag and denotes the number of observations that separate X_{t-j} and X_t. The values of ρ_j are in the range $[-1, 1]$. In most cases, ρ_j approaches zero as j increases. Qualitatively, ρ_j indicates the "similarity" in size between two elements spaced by j lags, which is different from zero only if the process has temporal dependence.

To understand the effect of autocorrelation on systems, we have simulated a closed system with two tandem queues and focused on transient performance measures at the autocorrelated resource. The model has two first-come first-served queues in tandem (e.g., a multi-tiered system), where a constant number of $N = 50$ jobs cyclically receive service from the two resources. Resource 1 has exponentially distributed service times with mean rate $\lambda = 1$; resource 2 has an autocorrelated service process that is represented by a Markov-modulated Poisson process with two states (i.e., an MMPP(2) process) [17]. Thus, queue 2 has temporal dependence in the service times and we are interested to investigate how this affects the overall system performance. We remark that MMPP(2) processes have been successfully used to model the service process under temporal dependent workloads in multi-tiered systems [12].

The MMPP(2) service process at queue 2 is a two-state continuous-time Markov chain where some jumps are associated to service completion events. Essentially, the chain spends in state i, $i \in \{0, 1\}$, a time that has exponential distribution $f(x) = \mu_i e^{-\mu_i x}$ and upon jumping out of i the system

completes job service with a certain probability, otherwise the job remains in service. The advantage of the MMPP(2) over other models is that an MMPP(2) can easily represent service times with high variability and temporal dependence [2]. The MMPP(2) considered in the case study is defined by the following parameters:

- state 0 has slow service rate $\mu_0 = 0.1$ job/sec;
- state 1 has fast service rate $\mu_1 = 12$ job/sec;
- the jumping probabilities between the two states $\{0, 1\}$ are selected such that the service times have mean service rate $\mu = 1$, squared coefficient of variation $CV^2 = 20$, and autocorrelation starting at lag-1 with $\rho_1 \approx 0.50$, see [5] for techniques for defining an MMPP(2) with predefined moments and lag-1 autocorrelation.

Figure 1 plots service times and queue-lengths at the autocorrelated resource 2 as a function of simulation time. Initially, the temporal dependent service process serves jobs with the fast rate μ_1 which results in a small queue-length. Just before point A, the service switches to the slow state with large service times, and jobs progressively accumulate in resource 2, until a new jump restores the fast service rate and flushes the jobs back to resource 1 (point B).

Let A_k, for $k \in \{0, 1\}$, be the *expected time duration* of the MMPP(2) process successively completing service in state k. Points C and D in Figure 1 illustrate the practical difference between strong and weak temporal dependence. Point C represents the outcome of jumps typically found in processes with *high* temporal dependence: the duration A_k of consecutively large (resp. small) service times is sufficiently long to allow a complete accumulation (resp. flushing) of the jobs. Point D shows instead the opposite effect which is easily found in a process with weak autocorrelation: the absence of temporal dependence leads to an irregularly alternating sequence of large and small service time that keeps the jobs in the system switching between the resources. Whenever behaviors of this type occur frequently, such as in models *without* temporal dependent service, the performance of the system improves compared to the highly-correlated case because the load on average is more equally shared by the resources [13]. In the highly-correlated case, instead, a single resource becomes highly-congested and slows down the entire system.

From the last observation, one can infer that the durations A_k have a strong impact on system performance and their online modification could be beneficial to control load. In a practical implementation, if we abstract the system in operation as here (i.e., evolving between "fast" and "slow" states), then we can use the idea of reducing A_k to better balance the load among the two queues, thus improving overall system performance. This is the underlying principle of the load control scheme defined below. In the next section, we introduce and study policies that alter the duration of the states associated with the longest processing times, thus blocking the performance effect of congestion periods, e.g., such as of the congestion starting at point C in Figure 1.

III. Autocorrelation-Guided Load Control

In this section, we propose a load control policy to improve the overall system performance. For the rest of the paper, we assume that load control by dropping requests is an acceptable practice for the application under consideration. For example, the MPEG video coding schemes store the necessary information to decode the video redundantly in multiple frames. Consequently, under heavy load, some of the frames can be dropped, up to a certain percentage, without compromising the overall quality of service perceived by the user. Furthermore, the quality of service perceived by the user depends on the device that is playing the digital video. If it is a low resolution device (such as a handheld), then the percentage of video frames that can be dropped without affecting the quality of viewing is higher than if the video is watched on a high-definition television.

The proposed load control policy is driven by autocorrelation that reduces the average durations of "slow" service periods. The duration of slow periods can be reduced by dropping longer (relatively to other) requests from the queue of the autocorrelated server. Note that reducing the duration associated to short requests also reduces autocorrelation and may thus improve performance. However, the performance impact of short jobs is typically small compared to that of long jobs and this makes the practice of dropping small jobs less interesting than for large jobs. Indeed, this drop does not necessarily correspond to an interruption of the service, but can be implemented in the real system in less invasive ways: e.g., an HTTP session processed by an overloaded Web server may be redirected to another server in the Web farm or have its latency artificially degraded to the benefit of the other concurrent sessions [21]. Our scheme is more effective than other methods, e.g., random drop, since we effectively forecast which jobs in the queue are long, thus become candidates to drop.

First, we present a static load control policy, called ALoC, which requires a priori knowledge of the autocorrelation function at the server process and of a user-defined parameter for controlling the load at the autocorrelated server. ALoC does *not* require a priori knowledge of the request size. Then, we present a related dynamic version, called D_ALoC, which also does not assume *any* a priori knowledge and dynamically adapts its load control parameter based on online measurement, policy targets, and statistical information of past workloads.

A. ACF-Guided Prediction

Dropping effectively the most harmful requests for performance depends on the prediction accuracy of future job sizes. Henceforth, we do *not* assume a priori knowledge of any job size, i.e., the system knows the size of a job only *after* it completes execution. The aim of the prediction is as follows: if size prediction is done effectively, long jobs can be accurately identified and removed from the queue so that the duration A_0 of the slow state decreases, yielding an improvement of the average response times due to the factors discussed before.

High-variance workload characteristics have been observed often in the literature [3], [8], [16], therefore our implicit assumption that jobs can be classified into long or short indeed applies to real systems. Here we assume that a job is *long* if its service time is larger than $\mu^{-1}(1+k\cdot CV)$, where μ^{-1} is the mean job size, CV is the coefficient of variation and k, e.g., $k=3$, is a given constant; the results presented throughout the paper are qualitatively similar for various choices of $k \geq 1$.

We now describe how we can use the autocorrelation function to predict job sizes. Let us assume that the last served job was long, i.e., its size was greater than $\mu^{-1}(1+k\cdot CV)$ for a given k, and suppose that we wish to forecast the size of the j-th job in queue (the job $j=1$ is the one immediately entering service after the long job just completed). If the service process has a positive ACF coefficient ρ_j, $0<\rho_j\leq 1$, then there is similarity in size between the completed long job and the j-th job in queue, therefore we forecast the j-th job as long[1]. Operationally, we cast a random number with uniform distribution in $[0,1]$, if the result is less than or equal to ρ_j, then we assume that the j-th job is long, otherwise it is short. That is, we assume that ρ_j is a measure of the conditional probability for a job to be large given that the last served job was large. Although ρ_j does not indicate the exact conditional probability that the j-th job is long given the current completed job is long, ρ_j quantitatively describes that conditional probability and thus can be used to effectively predict the size of jobs in queue.

A negative or zero ρ_j implies high probability that the j-th job significantly differs in size from the long job and therefore it is likely to be forecasted as short. Since in our analysis negative and zero autocorrelations lead to identical forecasting, we set $\rho_j=0$ whenever the measured lag-j autocorrelation is negative.

B. ALoC*: Static Version*

The high level idea of ALoC is as follows. After a long job is completed, the queue is scanned to find other long jobs probabilistically, according to their position j in the queue and the value of the corresponding autocorrelation coefficient ρ_j. All jobs that have been estimated as long are then dropped from the queue; indeed, if some jobs are known to be indispensable for the application they can be tagged as "undroppable" and be left in queue; we discuss this issue in Section IV. If we drop jobs based on autocorrelation then the estimated-short jobs remaining in queue are now more clustered, which elongates the duration A_1 at the expense of the congested state duration A_0 (i.e., the system should on average behave similarly to the part of Figure 1 before point C).

In order to control and maintain load reduction at a minimum, we introduce a queue length threshold QT for dropping requests, where $0<QT/N\leq 1$. Thus ALoC starts dropping requests only when the last completed request is long *and* the queue length at the autocorrelated server is higher than QT. Therefore, if the system is under-utilized and QT is not reached, then no request is dropped and all long requests are still served. Note also that since the policy is triggered only after a long job is executed, the policy avoids starvation of long jobs.

[1]See [2] for an overview of autocorrelated-based forecasting and other possible use of autocorrelation to predict future values in a time series.

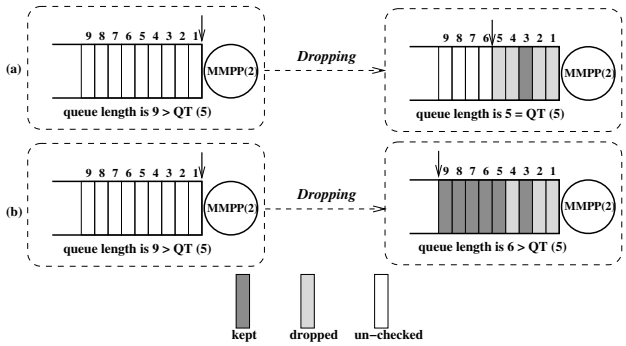

Fig. 2. **Illustration of ALoC's operations.** The most-recent served job is larger than $\frac{1}{\mu}(1+k\cdot CV)$ and the current queue length exceeds $QT=5$. Dark bars represent requests to be kept in the queue, light gray bars represent requests to be dropped, and blank bars represent requests yet to be considered by the policy. Figure 2(a) shows the policy stopping when four jobs have been dropped from the queue and the current queue length reaches QT. Figure 2(b) shows another possibility that the policy scans the entire waiting queue and only three jobs are estimated long ones and dropped.

We use an example to describe ALoC. In the example, $QT=5$ and there are 9 jobs waiting in the queue as shown in Figure 2. Upon completion of a long job, ALoC is triggered because the current queue length is greater than QT. ALoC starts to probabilistically predict the size of the j^{th} waiting job for $j=1,2,\ldots,9$. For instance, if $\rho_1=0.40$, then we interpret this value as a 40% probability that the job in position 1 is similar to the last completed job, i.e., it is a long job. We therefore cast a random number in $[0,1]$ and if the result is less than or equal to 0.40 the job in position 1 is dropped from the queue. A similar approach can be used to estimate the job service requirement for the j-th job in queue using the ρ_j autocorrelation coefficient, see light gray and dark bars in Figure 2 for an example of possible outcome of the forecasting. The policy has two stopping conditions: a first case is when enough estimated-long jobs have been dropped from the queue, and the current queue length has been reduced to the threshold value QT, see Figure 2(a). Alternatively, the policy may exhaust the waiting queue predicting that only estimated-short jobs wait in the queue with the current queue length still exceeding QT, see Figure 2(b). At the end of one round, the first job waiting in the current queue is admitted for service. ALoC is not triggered again before completion of another long job.

Figure 3 gives the pseudo-code for ALoC, which assumes a priori knowledge of the autocorrelation coefficients ρ_j, for $1\leq j<N$, of the service process, and the queue length threshold QT. In order to specify a fully autonomic load control policy, the controller must be able to estimate these values online. We introduce in Section III-D D_ALoC, the dynamic version of

1. initialize variables
  a. initialize index of the ACF queue: $i \leftarrow I$;
  b. initialize ACF values of the service stream at queue i: ρ_j for all $1 \leq j \leq N$;
  c. initialize threshold $QT \leftarrow R \cdot N$, for a given $0 < R \leq 1$;
2. for every job completion at queue i do
  a. check if service time of current request is long *and* the current queue length exceeds QT;
  b. if yes, start dropping
    I. initialize the index of jobs: $j \leftarrow 1$;
    II. for job j, generate a random number $s \in [0, 1]$
      if $s < \rho_j$, then assume the job is long and drop it;
      else assume the job is short and keep it;
    III. if the current queue length reaches QT,
      then go to step **2**;
      else $j \leftarrow j + 1$ and go to step **2-b-II**;
  c. else, go to step **2**;

Fig. 3. Description of ALoC. All input parameters are determined off-line.

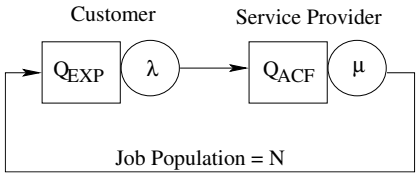

Fig. 4. A closed system with two queues Q_{EXP} and Q_{ACF} modeling a media streaming system.

ALoC which is able to do online estimation of all parameters.

C. Performance of ALoC

We use simulation to evaluate the performance of ALoC in a system of two tandem first-come-first-served (FCFS) queues Q_{EXP} and Q_{ACF} with mean service rates λ and μ, respectively, see Figure 4. This abstraction can be used to model a media streaming system, e.g., an Internet service provider streaming video and audio to customers such as personal computers, personal video recorders, game consoles, and mobile phones. We remark that the observations given in this section readily apply to systems with more than two queues.

Here, Q_{ACF} is the device with an autocorrelated service process, which is drawn from an MMPP(2) with mean rate $\mu = 1$ and squared coefficient of variation $CV^2 = 20$. Q_{EXP} is evaluated in two configurations:

- *Experiment 1*: Q_{EXP} is one order of magnitude faster than Q_{ACF};
- *Experiment 2*: Q_{EXP} is two orders of magnitude faster than Q_{ACF}[2].

That is, the service times of Q_{EXP} are exponentially distributed with mean rates $\lambda = 10$ and $\lambda = 100$ in *Experiment 1* and *Experiment 2*, respectively.

[2]Experiments with varying relative speed up to three and four orders of magnitude of the two devices yield qualitatively similar results and are not reported here due to limited space. The rationale behind considering different orders of magnitude in the service times is that this situation is typical when the service provider needs to read content from the disk.

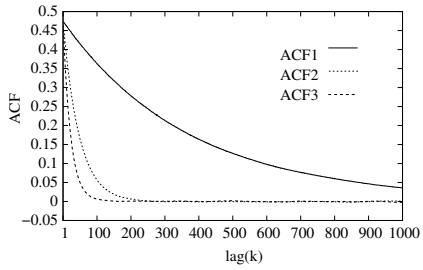

Fig. 5. The ACF of the service process that generates the autocorrelated flows in the system, where the service times are drawn from MMPP(2) with ACF_1, ACF_2 and ACF_3, respectively.

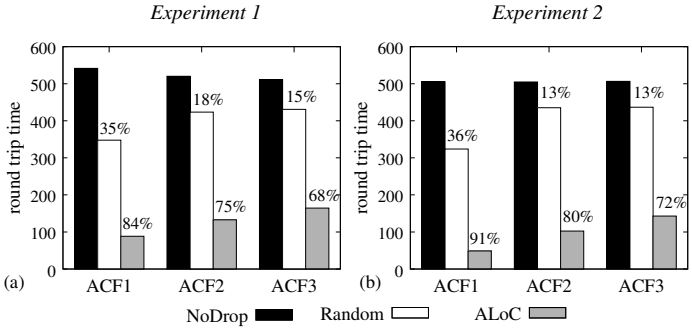

Fig. 6. Average response times of ALoC for *Experiment 1* and *Experiment 2*, with $N = 500$ and $QT = 490$ (98% of N). The drop ratios in the random and ALoC policies are 0.08, 0.10, and 0.13 for ACF_1, ACF_2, and ACF_3, respectively. The numbers above bars indicate the relative improvement over the NoDrop case.

In order to qualitatively analyze the effect of autocorrelation on policy performance, we also conduct experiments with MMPPs having different autocorrelation at Q_{ACF} for both *Experiment 1* and *Experiment 2*, but always such that they have the same mean, CV^2, and higher moments of the job sizes. We thus obtain three service processes with different autocorrelations: (1) ACF_1 with $\rho_1 = 0.47$ decaying to zero after lag 1500; (2) ACF_2 with $\rho_1 = 0.46$ decaying to zero beyond lag $= 240$; and (3) ACF_3 with $\rho_1 = 0.45$ decaying to zero beyond lag $= 100$. Figure 5 shows the ACF for the three service processes.

1) Comparison with Random Dropping: To compare the effectiveness of ALoC, we compare it with a policy where request drop is done *randomly*. The random policy continuously drops from the head of the waiting queue with probability set as same as the overall dropping ratio of ALoC. For ALoC we set $N = 500$ and $QT = 490$, i.e., 98% of N. Figure 6 presents the average response times for *Experiment 1* and *Experiment 2*. The relative *improvement* in round trip times is marked above each bar in the figure. Round trip times when all jobs are admitted without load control are plotted as a baseline comparison (NoDrop bars).

In the two experiments, drop ratios of ALoC are equal to 0.08, 0.10, and 0.13 for ACF_1, ACF_2, and ACF_3, respectively. Counterintuitively, the drop ratio for strong autocorrelated service process (e.g., ACF_1) is lower than that for the weak one, e.g., ACF_3. This is because with strong ACF, the prediction

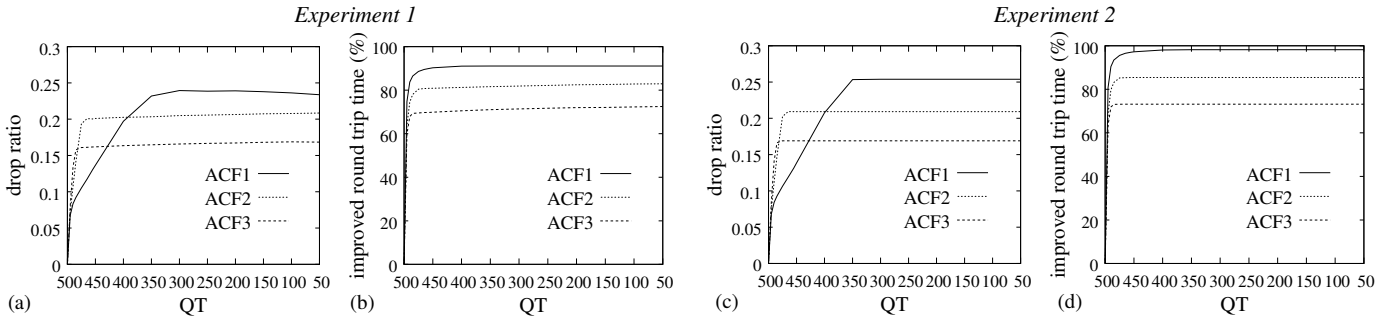

Fig. 7. Average drop ratio and average response time reduction achieved by ALoC as a function of QT for *Experiment 1* and *Experiment 2* with three MMPP(2) processes (i.e., ACF_1, ACF_2, and ACF_3) at Q_{ACF}. $N = 500$.

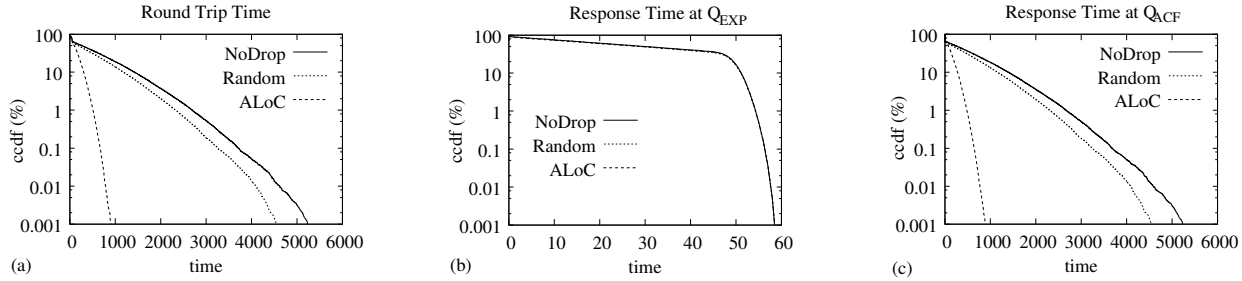

Fig. 8. CCDFs of response times for *Experiment 1* for the random and ALoC policies. Service times of Q_{ACF} have ACF_2, $N = 500$ and $QT = 490$. The drop ratio of both Random and ALoC is equal to 0.10.

of long jobs becomes more accurate, which increases the throughput of the autocorrelated queue and thus decreases the queue length. As the result of this, the trigger condition, i.e., the queue length being beyond QT, occurs less often. Consequently, the policy drops less jobs but improves the performance more with stronger ACF.

Figure 6 shows that across all experiments, ALoC dramatically improves expected response times compared to a random drop policy. Parsimonious selection of the request to be denied service results in significant improvements when compared to the random policy. In *Experiment 1* with ACF_1, the random policy results in a response reduction of about 35% from the baseline case. ALoC further reduces average response time by 84% relative to the baseline case. Performance trends persist for ACF_2 and ACF_3, but performance gains slightly reduce as the strength of ACF decreases. This can be explained by the fact that forecasting becomes less effective when the autocorrelations are smaller, and job size is thus harder to predict. Similar trends persist in *Experiment 2* where ALoC presents additional performance improvements as the speeds of the two devices differ now by two orders of magnitude. The higher the difference in the devices speed, the better the performance improvement of ALoC in comparison to dropping randomly.

2) Sensitivity to Queue Length Threshold QT: We quantify the performance effect of the pre-defined threshold QT used to trigger request dropping at the Q_{ACF} queue. We investigate the effectiveness of a choice of QT by computing the related average drop ratio and the relative improvement of response time. Reported statistics are only for those requests that complete work in both queues.

Figure 7 presents performance measures as a function of QT for *Experiment 1* and *Experiment 2* by using ALoC. The population in the model is set to $N = 500$. QT ranges from 100% of N, i.e., no drop since ALoC is never triggered, to 10% of N, i.e., we drop requests when the queue length in Q_{ACF} is equal to 50. From the figures we see that as QT decreases, the drop ratio increases quickly (see Figure 7 (a) and (c)), but there is a point beyond which the drop ratio stabilizes. This happens because the smaller the QT, the larger the proportion of large jobs that are denied service at Q_{ACF}. When QT becomes too small, the policy becomes very aggressive: most long jobs are dropped and only few long jobs remain in the queue to be dropped. Therefore, the drop ratio stabilizes because the queue is almost empty of long jobs. Across both experiments, the position of the knee of the drop ratio curve depends on the strength of the autocorrelation function. The stronger the autocorrelation, the lower the value of QT for which the knee appears.

The performance effect as a function of QT values is illustrated in Figure 7 (b) and (d). The plots show that excellent performance improvements can be achieved by triggering ALoC infrequently with large QT values, which also results in desirable smaller drop ratios. In Figure 7 (b) and (d), a large QT equal to 490 results in dramatic performance improvements while containing the drop ratio at a minimum across all experiments.

For completeness, we have also conducted sensitivity analysis under different job populations, e.g., $N = 100$ and $N = 300$. Our results can be summarized as follows. Drop ratios

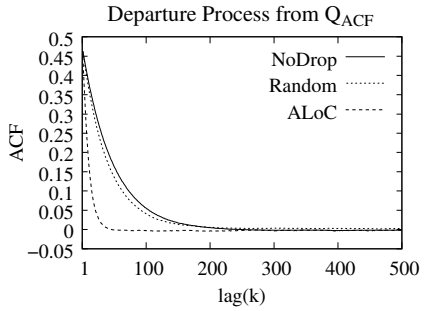

Fig. 9. ACF of the departure process of Q_{ACF} for *Experiment 1*. $N = 500$, the autocorrelation of the service process is ACF_2, and $QT = 490$. The drop ratio of random and ALOC is 0.10.

and relative performance gains with different populations are qualitatively the same as those for $N = 500$. Drop ratios are lower in less populated models while relative gains in response times remain high.

3) Round Trip Time Distribution: We analyze the tail performance and plot in Figure 8 the complementary cumulative distribution function (CCDF) of round trip times and of response times at Q_{EXP} and Q_{ACF} for *Experiment 1* with ACF_2 . Results for ACF_1 and ACF_3 are remarkably similar to those reported in this figure. The figure shows that ALOC significantly improves the tail of the response times at Q_{ACF} and consequently the response times. The tails of response times at Q_{EXP} of all three policies are almost identical.

Figure 9 also depicts the ACFs in the departure process of Q_{ACF} (i.e., arrivals to Q_{EXP}). The random policy achieves a small reduction in the autocorrelation function compared to the original one (labeled as "NoDrop" in the figure). ALOC's ability to selectively deny service of jobs in the queue that cause autocorrelation is shown in the figure: the departure process curve that corresponds to this policy shows autocorrelation that is significantly reduced.

D. **D_ALOC***: Dynamic Policy*

This version of the policy does not require any a priori knowledge of either autocorrelation coefficients or queue length threshold QT. D_ALOC computes the autocorrelation coefficients online, allowing for changes in the workload characteristics over time and self-adjusts QT such that target performance parameters are met.

For each server, D_ALOC evaluates its mean service time, its coefficient of variation, and the autocorrelation coefficients of the service process when a job is completed at that particular server, using a modified version of Welford's one-pass algorithm [22]. The definition of ACF at lag j given in Eq. 1 can be rewritten as follows:

$$\rho_j = (E[X_{t-j}X_t] - E[X_{t-j} + X_t] \cdot \mu^{-1} + (\mu^{-1})^2)/\sigma^2, \quad (2)$$

where μ^{-1} and σ^2 are respectively mean and variance of the sequence and

$$\begin{aligned} E[X_{t-j}X_t] &= E[X_{t-j-1}X_{t-1}] \\ &\quad + \frac{X_{t-j}X_t - E[X_{t-j-1}X_{t-1}]}{t} \\ E[X_{t-j} + X_t] &= E[X_{t-j-1} + X_{t-1}] \\ &\quad + \frac{X_{t-j} + X_t - E[X_{t-j-1} + X_{t-1}]}{t}. \end{aligned}$$

Once a job with service time of X_t is completed, the values of $E[X_{t-j}X_t]$ and $E[X_{t-j} + X_t]$ are updated and autocorrelation ρ_j is recalculated using Eq. 2. If the autocorrelation coefficients in a specific server are positive, then D_ALOC determines that this server is the source of autocorrelation in the traffic flows of the entire system. Consequently, the load reduction is triggered at that server.

1. initialize threshold $QT \leftarrow N$;
2. initialize the maximum allowable drop ratio D;
3. for every C requests do
  a. upon each job completion at queue i,
    I. calculate ACFs using Eq. (2): ρ_j for all $1 \leq j \leq N-1$;
    II. if the service process at queue i is autocorrelated, then drop using the same scheme of ALOC;
  b. at the end of an updating window of C requests
    I. calculate current drop ratio d and compare with D;
    II. adjust QT using Eq. (3);

Fig. 10. Description of D_ALOC. *All* policy parameters are computed online.

To dynamically adjust QT, we use an updating window of C requests that have been served. In the experiments presented here C is set to 3000. The value of QT is initialized to N. For every batch of C requests, D_ALOC compares the current request drop ratio d with the maximum allowable drop ratio D. If the current drop ratio exceeds D, then QT is increased to reduce the frequency of dropping requests. If the drop ratio d is below D, then QT is reduced to drop requests more aggressively. The following equation illustrates how QT changes by a value that is proportional to the difference between the drop ratio d and the allowable drop ratio D:

$$QT = \begin{cases} QT + (N - QT) \cdot \frac{d-D}{1.0-D} & \text{if } d > D \\ QT - (QT - 0) \cdot \frac{D-d}{d-0.0} & \text{if } d \leq D \end{cases} \quad (3)$$

Upon updating QT, the new threshold for the next C requests in the autocorrelated server. Figure 10 gives the pseudo-code for D_ALOC.

E. Performance of D_ALOC

In this section, we evaluate the effectiveness of D_ALOC. The simulation environment is the same as in Section III-C. For all experiments presented here, we set the maximum allowable drop ratio equal to 0.06, 0.08, 0.10, or 0.13. Figure 11 presents the average response times as a function of drop ratio in *Experiment 1* and *Experiment 2* under both D_ALOC and ALOC policies. Round trip times when all jobs are admitted are also plotted as a baseline comparison. Here, ALOC is parameterized such that the ideal QT is selected

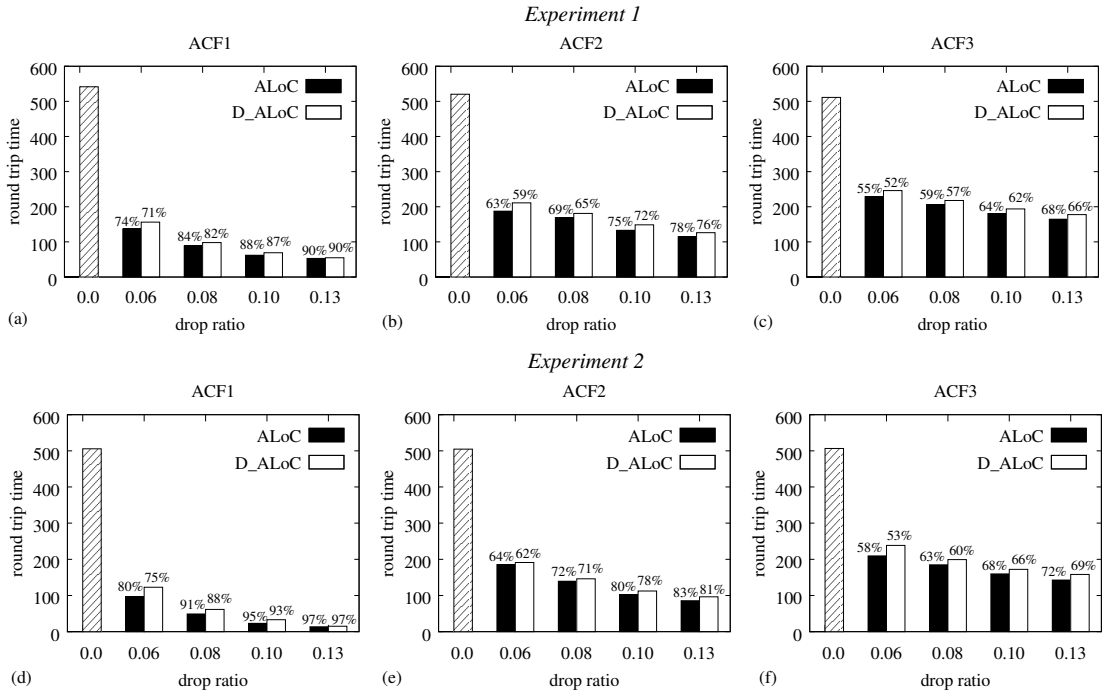

Fig. 11. Average response times under drop ratios of 0.0, 0.06, 0.08, 0.10, and 0.13, for *Experiment 1* and *Experiment 2*. Service times of Q_{ACF} are drawn from three different MMPP(2)s (i.e., ACF_1, ACF_2, and ACF_3). $N = 500$. The number above bar are the relative improvements over the NoDrop case.

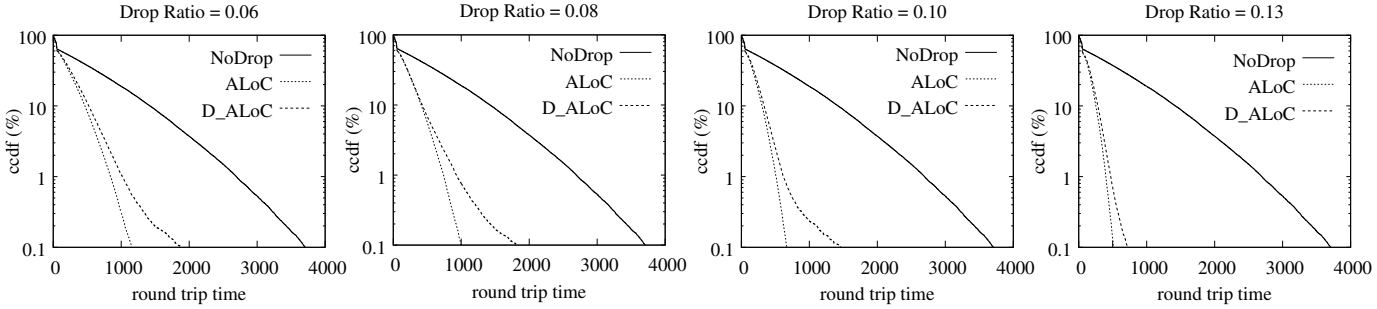

Fig. 12. CCDFs of response times under drop ratios of 0.0 (i.e., no drop), 0.06, 0.08, 0.10, and 0.13, for *Experiment 1*, where the service times of Q_{ACF} are drawn from ACF_2. $N = 500$.

to achieve the pre-defined drop ratio while achieving best performance. At the beginning of the simulation, D_ALoC initializes $QT = 500$ (i.e., no drop), but it gradually changes this value such that the average drop ratios are maintained below the corresponding pre-defined allowable drop ratio.

The experiments depicted in Figure 11 indicate that D_ALoC's performance is very close to ALoC's. This means that D_ALoC is truly effective, especially because for each ALoC bar in Figure 11, the value of QT is selected by exhaustive searching so that ALoC achieves the best response values.

Figure 12 illustrates the CCDF of response times under drop ratios of 0.0 (i.e., no drop), 0.06, 0.08, 0.10, and 0.13 for *Experiment 1* with ACF_2 in the service stream at Q_{ACF}. The figure clearly shows that both ALoC and D_ALoC significantly improve the tail of response times. The tail of D_ALoC is close to that of ALoC, and the gap between these two tails diminishes as the drop ratio increases. In summary, Figures 11 and 12 argue for D_ALoC's effectiveness and robustness with respect to different autocorrelation strengths in the service process of Q_{ACF}, different target drop ratios, and relative speeds of Q_{EXP} and Q_{ACF}.

IV. Trace Driven Evaluation

The majority of Internet-based media streaming systems can be modeled as a closed queueing system, see Figure 4. In such a model, the first queue represents the device which receives the streaming media, e.g., a personal computer and other consumer electronic devices and the second queue represents the server that has stored the media content, e.g., movies, songs and games.

In this section, we use actual traces measured at the disk level of a streaming system to evaluate how ALoC and D_ALoC perform in a practical setting. The traces record,

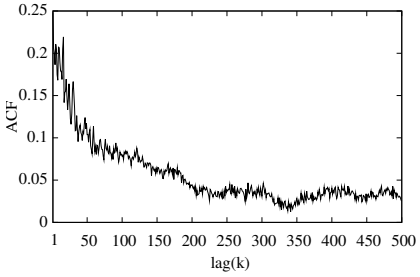

Fig. 13. The ACF of the service times at a streaming device.

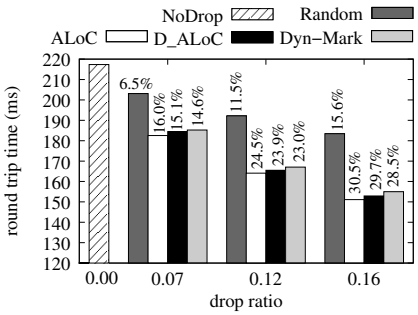

Fig. 14. Average performance response times when 0% (no dropping), 7%, 12%, and 16% of the work in the second server (disk) is dropped. $N = 200$. The numbers above bars indicate the relative improvement over the NoDrop case.

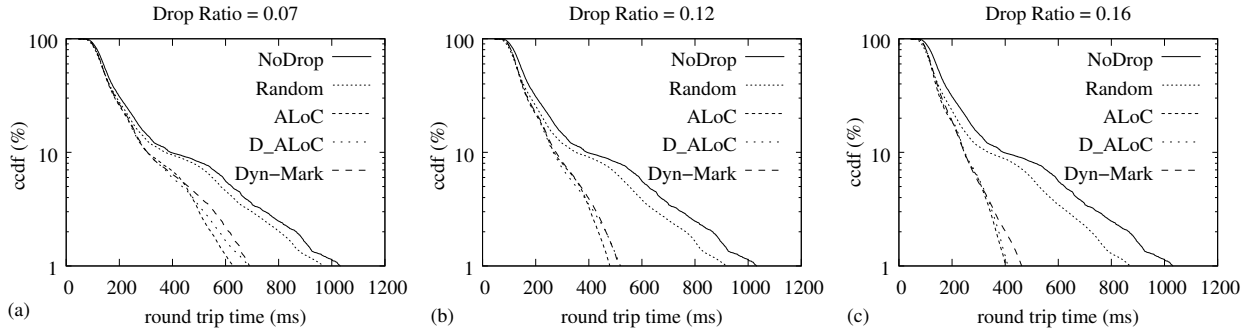

Fig. 15. CCDFs of response times using the real traces. $N = 200$.

in high resolution, both arrival and departure times of each request. Further details on these traces and their representativeness can be found in [16].

The mean service time recorded in the trace is 1.09 ms and CV is equal to 2.47. Figure 13 presents the autocorrelation function for the disk service process in this trace. At the first queue in our model, service times are drawn from an exponential distribution with mean service time equal to 0.01 ms, i.e., the first server is two orders of magnitude faster to view the content than the server that reads the content from the disk. The population N in the system is set to 200 and the sample space is equal to 1,043,259 requests. We remark that, for increased values of the population N, the autocorrelation of disk request sizes would be the same since in queueing models the service process is commonly assumed to be independent of N.

To investigate policy robustness, we add an additional restriction by marking some requests as "undroppable". In particular, we focus on trace data where the transmitted files are MPEG video streams. MPEG video streams compress raw frames specifically into three kinds of pictures: (1) I(ntra-coded)-pictures, which are independent of others, (2) P(redictive-coded)-pictures, which depend on the previous I- or P- pictures for being displayed correctly, and (3) B(idirectionally predictive-coded)-pictures, which need the information from the previous and the following I- or P-pictures for motion compensation [10]. I-pictures are the most important pictures and thus cannot be dropped, while a limited drop of a P- or B- pictures is acceptable. We therefore investigate the effectiveness of D_ALoC under the restriction that some requests are undroppable (i.e., I-pictures). This variation of D_ALoC is called "Dyn-Mark".

We show response times using the actual traces as a service process of the streaming server for random, ALoC, D_ALoC, and "Dyn-Mark" policies. Figure 14 plots the average response time for the various policies when 7%, 12%, and 16% of the total requests are dropped. Additionally, ALoC is tuned such that it achieves its best performance for the target drop ratio. The relative performance improvement of the various policies compared to a no drop policy is consistent with the results presented in the previous section: as the drop ratio increases, the response times significantly decrease. D_ALoC self-adjusts its configuration parameters and achieves closely as good response times as the carefully tuned ALoC. The restriction of dropping certain requests in Dyn-Mark results in a slight degradation in performance improvements, but nevertheless significant gains in comparison to no drop.

Figure 15 plots the tails of the response time distribution for all policies. Results are consistent with those reported in the synthetic trace, further arguing for the robustness of D_ALoC even with drop restrictions.

V. Conclusions and Future Work

In this paper we have proposed ALoC and D_ALoC, two autonomic load control policies that use for the first time autocorrelation measured in service times as a criterion

to selectively drop queued requests and control load in a distributed system. Using autocorrelation, both policies are able to effectively guess the future service requirements of incoming jobs at a server and drop the load according to this forecasting information.

Using simulations, we have shown that ALoC and D_ALoC are able to reduce end-to-end system response times for different workload intensities, levels of autocorrelation, and target drop ratios. Experiments on synthetic traces show that the response time improvement of ALoC and D_ALoC typically varies between 50% and 80%. On a real trace where some requests are marked as high priority requests and at the extreme case as "undroppable", both policies are still very effective, with a response time improvement between 15% and 30%. These results promote ALoC and D_ALoC as easy-to-implement policies for load control in autonomic systems. In the future, we plan to study the behavior of systems where multiple resources have autocorrelated service times.

Acknowledgments

This work is supported by NSF grants CNS-0720699 and CCF-0811417.

References

[1] A. M. Adas, A. Mukherjee. On resource management and QoS guarantees for long range dependent traffic. In *Proc. INFOCOM '95*, pages 779–787, 1995.

[2] J. Beran. *Statistics for Long-Memory Processes*. Chapman & Hall, New York, 1994.

[3] A. B. Bondi, W. Whitt. The influence of service-time variability in a closed network of queues. *Perform. Eval.*, 6:219–234, 1986.

[4] L. Cherkasova, P. Phaal. Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites. *IEEE Trans. Computers*, 51(6):669–685, 2002.

[5] J. Diamond, A. Alfa. On approximating higher-order MAPs with MAPs of order two. *QUESTA*, 34:269–288, 2000.

[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, Z. Liu, C. H. Xia, L. Zhang Load shedding and distributed resource control of stream processing networks In *Proc. of IFIF Performance 2007*, Perform. Eval. 64(9-12), 1102-1120, 2007.

[8] S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, and E. L. Miller. Self-similarity in file systems. In *Proc. of SIGMETRICS*, pages 141–150, 1998. ACM.

[9] H. Li, M. Muskulus. Analysis and modeling of job arrivals in a production grid. *SIGMETRICS PER*, 34(4):59–70, 2007.

[10] *ISO/IEC 13818: Generic coding of moving pictures and associated audio (MPEG-2)*.

[11] D. Menasce, V. Almeida. *Capacity Planning for Web Performance: Metrics, Models, and Methods*. Prentice Hall, 1998.

[12] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *ACM/IFIP/USENIX 9th International Middleware Conference (Middleware'08)*, Leuven, Belgium, 2008.

[13] N. Mi, Q. Zhang, A. Riska, E. Smirni, and E. Riedel. Performance impacts of autocorrelated flows in multi-tiered systems. *Perform. Eval.*, 64(9-12):1082–1101, 2007.

[14] R. O. Onvural and H. G. Perros. Equivalencies between open and closed queueing networks with finite buffers. *Perform. Eval.*, 9:263–269, 1989.

[15] V. Paxson, S. Floyd. Wide-area traffic: the failure of Poisson modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244, 1995.

[16] A. Riska, E. Riedel. Long-range dependence at the disk drive level. In *QEST '06: Proc. of the Third International Conference on the Quantitative Evaluation of Systems*, pages 41–50, Riverside, CA, USA, 2006. IEEE Computer Society.

[17] T. Robertazzi. *Computer Networks & Systems: Queueing Theory and Performance Evaluation*. Springer-Verlag, 1990.

[18] B. Schroeder, G. A. Gibson. Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you? *ACM Trans. Storage*, 3(3):8, 2007.

[19] A. Thummler, M. Telek. A Novel Approach for Phase-Type Fitting with the EM Algorithm. *IEEE T. Depend. Secur. Comput.*, 3(3):245–258, 2006.

[20] B. Urgaonkar , G. Pacifici, P. Shenoy, M. Spreitzer, A. Tantawi. An analytical model for multi-tier internet services and its applications. *Proc. of SIGMETRICS*, 291–302, 2005.

[21] M. Welsh , D. E. Culler, E. A. Brewer. An Architecture for Well-Conditioned, Scalable Internet Services *Proc. of the 18th ACM symposium on Operating systems principles (SOSP)*, 230–243, 2001.

[22] B. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4:419–420, 1962.