ASIdE: Using Autocorrelation-Based Size Estimation for Scheduling Bursty Workloads

Ningfang Mi, *Member, IEEE,* Giuliano Casale, *Member, IEEE,* and Evgenia Smirni *Member, IEEE*

***Abstract*—Temporal dependence in workloads creates peak congestion that can make service unavailable and reduce system performance. To improve system performability under conditions of temporal dependence, a server should quickly process bursts of requests that may need large service demands. In this paper, we propose and evaluate ASIdE, an Autocorrelation-based SIze Estimation, that selectively delays requests which contribute to the workload temporal dependence. ASIdE implicitly approximates the shortest job first (SJF) scheduling policy but *without* any prior knowledge of job service times. Extensive experiments show that (1) ASIdE achieves good service time estimates from the temporal dependence structure of the workload to implicitly approximate the behavior of SJF; and (2) ASIdE successfully counteracts peak congestion in the workload and improves system performability under a wide variety of settings. Specifically, we show that system capacity under ASIdE is largely increased compared to the first-come first-served (FCFS) scheduling policy and is highly-competitive with SJF.**

***Index Terms*—Temporal dependence, delay-based scheduling, no-knowledge scheduling, FCFS scheduling, SJF scheduling**

I. Introduction

Burstiness and temporal dependence in workloads are often found in multi-tier architectures, disk drives, and grid services [15], [20], [25]. These features are responsible of dramatic degradations of the system performability (i.e., performance and availability) as they create peak congestion that can make service unavailable. Under bursty conditions, system availability is strongly dependent on the response taken to counteract workload peak congestion and on the adoption of efficient schemes that address burstiness.

The characterization and the definition of remedies for temporal dependence congestion effects have been exhaustively studied in networking, leading to the development of accurate models of autocorrelated traffic processes (e.g., Markov Modulated Poisson Processes (MMPPs) and fractional Brownian motion) [32], and to measurement-based load-control schemes for network availability under rapidly changing flows [9]. The schemes in [9], [32] are based on fitting distributions to data. However, the fitting problems can become very challenging, especially when one wants to find a fit to measured higher moments and temporal dependence of the data. Even if one can solve such fitting problems, it is still difficult to know how to use the fitted distributions in scheduling. Hence, these schemes unfortunately cannot be easily applied to systems. In this paper, we fill this gap by addressing the lack of effective measurement-based schemes to maintain performance and availability in systems with bursty workloads. We focus on systems with finite buffers and/or admission control. In addition, we assume that work reduction techniques such as dropping jobs [19] cannot be applied, i.e., processing of the entire workload is mandatory. This assumption makes the problem more challenging. We also remark that the terms "burstiness", "temporal dependence" and "serial correlation" are interchangeably used in the text.

The shortest job first (SJF) scheduling is well-known from classic scheduling theory [26] for minimizing the average completion time of all jobs if knowledge of *exact job sizes* is available to the scheduler. However, the optimality of SJF depends on a priori knowledge of job service times, which is not practical in systems where job service times are only known after the jobs are completed. The purpose of this paper is to present a new autocorrelation-based size estimator called ASIdE that effectively estimates (i.e., differentiates) long/short job service times using the burstiness characteristics of the service process and implicitly approximates SJF by delaying jobs that are expected to be long.

We show that ASIdE predicts requests as short or long based on burstiness in the service process. ASIdE is based only on measurement of past workload (e.g., service times of completed jobs), leveraging on the structure of temporal locality to forecast the size of upcoming requests and defining self-adjusting criteria to delay requests that it deems as large. By selectively delaying requests that contribute to temporal locality, ASIdE significantly improves system throughput, which results in increasing the amount of requests that a server can process at a given time and deals effectively with congestion conditions. We interpret the significant performance improvement as an outcome of the burstiness reduction in service process of the resource where requests are delayed. More importantly, in the case where there is a series of servers that a job needs to be served before completion (e.g., [20]), decreasing the source of burstiness at one server reduces burstiness propagation [20] in the flow of the entire network of servers, which results in generalized throughput improvement.

Experimental results show that ASIdE can increase throughput under temporally dependent workloads, without service rejection, while delaying only a small fraction of jobs. ASIdE

N. Mi is with Northeastern University, Boston, MA, USA; E-mail: ningfang@ece.neu.edu.

G. Casale is with Imperial College London, London, UK; E-mail: g.casale@imperial.ac.uk.

Evgenia Smirni is with College of William and Mary, Williamsburg, VA, USA; E-mail: esmirni@cs.wm.edu.

Manuscript received Sept. 8, 2010; revised Jan. 6, 2011, Aug. 14, 2011, and Jan. 7, 2012. The associate editor coordinating the review of this paper and approving it for publication was Joe Hellerstein.

This work was supported by the National Science Foundation under grants CNS-0720699 and CCF-08114171. A preliminary version of this paper appeared in [22].

dramatically increases system capacity compared to FCFS scheduling and works comparably to SJF despite the fact that it does not require any a priori knowledge of future workload. Sensitivity analysis with respect to different workload characteristics shows that ASIdE is effective and robust. Using two real traces, ASIdE is shown to be also effective with real-world workloads. The remainder of the paper presents our results in detail.

II. Delay-Based Scheduling Policy: ASIdE

In this section, we introduce ASIdE, a delay-based scheduling policy that improves performance and availability in systems with temporally dependent workloads. The basic idea behind ASIdE is to implicitly approximate the SJF scheduling policy but *without* requiring any knowledge of job service times. ASIdE uses the measured temporal dependence in the service times to estimate the job service time of upcoming jobs. Once reliable estimates of the job service times are available, we delay large jobs by putting them at the tail of the queue up to a fixed number of times. As a result, long jobs will be more likely served after most short jobs are served. Estimated short jobs are instead never delayed by ASIdE.

Summarizing, the ideas that ASIdE is based on are: (1) to approximate the behavior of the SJF scheduling discipline by proper use of job delaying; and (2) to estimate the expected service times of the jobs waiting in queue from the temporal dependence in the job service process. We stress that ASIdE does *not* assume any a priori knowledge of the service time of any of the enqueued jobs. The system knows the exact service time received by a job only *after* the job completes execution. Estimation of service times for upcoming jobs is based only on past history. ASIdE also incorporates mechanisms to avoid job starvation. In the following subsections, we present the policy in detail.

A. Forecasting Job Service Times

The effectiveness of the proposed policy depends on the accuracy of forecasting job service times. If prediction is done effectively, then long jobs to be delayed can be accurately identified and ASIdE performs optimally. We first present the forecasting approach which is specifically tailored to temporally dependent workloads.

Exploiting Service Time Variability: ASIdE's service time forecasting relies on two system aspects: service time variability and temporal dependence. Concerning the former, we leverage on the fact that service time distributions found in systems are typically characterized by high variance and heavy tails [2], [25] and therefore the differentiation between small and large service times can be performed effectively and used to improve performability. Rather than using a constant value, ASIdE uses a large-job threshold (LT),

$$LT = \mu^{-1} + k \cdot \sigma, \tag{1}$$

where μ^{-1} is the mean service time at the resource, σ is the standard deviation of service times, and $k >= 0$ is a constant determined online[1]. If a job service time is greater than LT, then ASIdE regards the job as "long" (also referred throughout the paper as "large"). Otherwise, ASIdE classifies it as "short". Note that the policy can successfully measure the parameters for computing LT in an online fashion, i.e., the mean μ^{-1} and the standard deviation σ of the service times are continuously updated in ASIdE using Welford's one-pass algorithm [31].

Exploiting Temporal Dependence: Given a classification of jobs into long and short, the next step is to effectively forecast whether the jobs currently in the queue are long or short. To this end, we exploit the structure of temporal dependence in order to make an educated "guess" whether a job in the queue is long or short. We assume that the scheduler is able to measure correctly the service times of jobs that are completed by the server; this is readily available in most systems. Let T be the time instant in which a forecasting decision is needed. This instant always corresponds to the departure instant of a *long* job leaving from the queue. Also assume that during the period $[T - T_W, T]$, the system has completed n jobs with service times $S_1, S_2, \ldots, S_n$, where T_W, $0 \leq T_W \leq T$, is a window monitoring past scheduling history. A forecasting decision is always done at the departure instant of a *long* job leaving from the queue.

ASIdE's forecasting approach is based on the estimates of the conditional probabilities:

$$P[L|L]_j = P[S_{t+j} \geq LT | S_t \geq LT] \tag{2}$$

$$P[S|L]_j = P[S_{t+j} < LT | S_t \geq LT] = 1 - P[L|L]_j \tag{3}$$

which are computed using the measured service times S_t for $t = 1, \ldots, n - j$. Here j is called the lag of the conditional probability and denotes the distance between the job service completions. Given that the last completed job is long, $P[L|L]_j$ measures the fraction of times that the j-th job that has arrived after a long job is also long; similarly, $P[S|L]_j$ estimates how often the j-th job is instead a short job. Using these estimates, we forecast that the j-th arrival after the last completed job is going to receive large service time if

$$P[L|L]_j \geq P[S|L]_j, \tag{4}$$

i.e., if the empirical probability that the j-th arrival being long is higher than being short. ASIdE is triggered only when the last finished job is long. Since we focus on the systems with finite buffer or with constant population N, we *only* make use of the conditional probabilities $P[L|L]_j$, for $1 \leq j < N$.

The algorithm in Figure 1 describes how to online calculate the conditional probabilities based on past history during the last monitoring window T_W. In each monitoring window, the system completes exactly W requests; in the experiments presented here, we set $W = 10,000$. Consider as an example two classes of job service times, i.e., $C = 2$ for large and small ones. Upon each job completion, ASIdE updates the number of large jobs completed during the monitoring window and updates the conditional probabilities $P[L|L]_j$ and $P[S|L]_j$ for $1 \leq j < N$ by scanning N jobs in two directions (i.e.,

[1]We remark that Eq.(1) is used here to classify large service times from small ones because the standard deviation indicates how far values are from the mean [13].

arriving before and after the completed job) as follows. If the last completed job is long, then ASIdE scans the N jobs that arrived *before* the completed job and increases the number of $L|L$ (resp. $S|L$) pairs at lag j by 1 if the j-th job is completed and its job size is long (resp. short). The N jobs that arrive *after* the last completed job are scanned as well. If the j-th job is completed and its job size is long, then ASIdE increases the number of $L|L$ (resp. $S|L$) pairs at lag j by 1 if the last completed job is long (resp. short). At the end of the monitoring window, ASIdE recalculates $P[L|L]_j$ (resp. $P[S|L]_j$) using the ratio of the number of $L|L$ (resp. $S|L$) pairs at lag j and the total number of jobs that are long.

Illustrating Example. Figure 2 shows an example that builds intuition on the tight relation between our forecasting approach and temporal dependence in service times. Figure 2 illustrates conditional probabilities in a sequence *without* temporal dependence and *with* temporal dependence. The two sets of figures (i.e., Figure 2(a)-(b) and Figure 2(c)-(d)) compare the conditional probabilities $P[L|L]_j$ and $P[S|L]_j$ with the similarly defined $P[S|S]_j$ and $P[L|S]_j$ which assume that the last completed job was short. As one can see, useful information that can be exploited in prediction is found only in Figure 2(II) where there is temporal dependence in the process. Figure 2(I), instead, shows that without temporal dependence the probability of the next service time being small or large neither depends on the lag j nor on the type of job that just completed. The opposite is observed in Figure 2(II), where the lag-j probability of the next job being short or long strongly depends on the size of the completed job. In addition, we also see that the conditional probability of having large jobs within the next nine arrivals after a long job is significant and ranges from 0.65 to 0.35, see Figure 2(d). Similarly, the probability of having a short job after another short job is very large, see Figure 2(c).

Summarizing, workloads for which service times are independent and no temporal locality exists, cannot be used to forecast future service requirements; conversely, temporally dependent service processes can be useful in predicting future service requirements. We exploit in ASIdE this property to approximate the behavior of SJF scheduling.

B. The Delaying Algorithm: ASIdE

We now describe ASIdE in detail. For presentation simplicity, we assume here that the large threshold LT, see Eq.(1), that is fundamental for forecasting is given; in the next subsection, we present how ASIdE self-adjusts LT on-the-fly, i.e., no a priori knowledge of LT is required. We also assume two classes of service times, i.e., $C = 2$. Figure 3 gives the pseudo-code of ASIdE.

Upon the completion of a long job, ASIdE scans the entire queue and predicts the size of the j-th queued job by using the conditional probabilities as described in the previous subsection. If the j-th job is estimated as long, then ASIdE marks it as such. All jobs that are marked long are delayed by moving them to the end of the queue. After all jobs in the queue have been examined and long jobs have been delayed, ASIdE admits the first job in the queue for service. Delaying is triggered again only after completion of another long job.

Additionally, during each monitoring window T_W, ASIdE updates the information of job service times after each completion of a job and recalculates conditional probabilities at the end of the window as described in Figure 1. The obtained conditional probabilities are then used to estimate the service times for the upcoming jobs in the next monitoring window. ASIdE starts its estimation process *after* the first monitoring window. As a result, all jobs completed in that window are only used as a statistically significant sample for obtaining a good estimation of the conditional probabilities.

ASIdE "reshuffles" all jobs in the queue based on their anticipated service times; the order of the jobs in the service process is therefore altered (attempting to approximate SJF scheduling) and this modifies both the throughput at the queue and the serial correlation of the process. Concerning the latter, we point to [1] for an accurate analysis of the effects of shuffling on serial correlation. To avoid starvation of long jobs, we further introduce a *delay limit* D that caps the maximum number of times a single job can be delayed. When the number of times a job has been delayed is more than D, the policy does not delay this job any longer and allows it to wait for service in its current position in the queue.

ASIdE does not re-forecast the length of a job whose service time has been already forecasted to be long. Once a job has been marked as long it remains as such and is never forecasted as short in successive activations of ASIdE. The same property also holds for short jobs. Once a job is estimated as long and delayed, ASIdE cannot use the conditional probabilities to re-forecast its length because the order of the jobs in the service process has been altered. For example, if we have the first arriving job to be long and delayed by ASIdE, then the second arriving one now becomes the first job waiting in the queue and is admitted for service. In the case that the served job (i.e., the second arriving one) is actually long, one round of delaying is triggered, see Step 4 in Figure 3. However, as the conditional probabilities on the sequence of jobs in the queue are computed according to their arrival indexes, ASIdE can only estimate the size of the jobs arrived after the second job completion using conditional probabilities $P[L|L]_{lag} \geq P[S|L]_{lag}$, given that the completed one is long.

C. Self-Adjusting the Threshold LT

In this subsection, we discuss how ASIdE adjusts the threshold LT, aiming at controlling the magnitude of the delaying, in order to strike a good balance between being too aggressive or too conservative. Intuitively, when the threshold LT is too large, the policy becomes conservative by delaying few long jobs and the performance improvement is then negligible. Conversely, when LT becomes too small, more jobs (even short ones) are delayed and throughput is reduced. Therefore, the choice of an appropriate large threshold LT is critical for the effectiveness of ASIdE.

In ASIdE, the computation of LT is a function of the monitoring window T_W as well. The algorithm in Figure 4 describes how the threshold LT is dynamically adjusted in an

on-line fashion. At the end of a monitoring window T_W, we update LT while keeping as upper and lower bounds for its value the 90th and the 50th percentiles of the observed service times within T_W, respectively. Indeed, whenever specific information on the workload processed by a system is available, these values can be increased or decreased according to the characteristics of the workload. The threshold LT is updated by assuming that the value of the conditional probability $P[L|L]_j$ at some large lag j is representative of the overall tendency of the system to delay jobs.

In the implementation considered in the paper, we adjust the parameter k which defines $LT = \mu^{-1} + k \cdot \sigma$ with step adj according to the following scheme. Let Q be the current queue-length at the server. We evaluate $P[L|L]_j$ for the large lag $j = \lfloor Q/2 \rfloor$ and if $P[L|L]_j \geq P[S|L]_j$, then ASIdE is assumed to be too aggressive, since it may delay at the next round up to $\lfloor Q/2 \rfloor$ jobs. In this case, we set $k = k + adj$, which reduces the number of jobs identified as long. As a result, we can avoid half of total requests waiting in the queue to be delayed. A similar procedure is done for the case $j = \lfloor Q/10 \rfloor$, where if $P[L|L]_j < P[S|L]_j$, we then conventionally assume that ASIdE is too conservative by delaying less than 10% of jobs in the queue; in this case we set $k = k - adj$ which increases the number of jobs estimated as long. Since delaying jobs in an aggressive way may achieve worse performance than in a conservative way, we here set $j = \lfloor Q/10 \rfloor$ instead of $\lfloor Q/2 \rfloor$ to guarantee at least 10% of queued requests to be delayed. Extensive experiments suggest that the LT online algorithm is consistently stable.

A typical example is shown in Figure 5. When an appropriative value of adj is chosen, the large threshold LT quickly converges to a stable value after some monitoring windows, e.g., $adj = 0.01$ and 0.5 in plots (a) and (b). On the other hand, if the value of adj is too large, then the large threshold LT cannot converge to a stable value but instead oscillates between two extreme values, see plots (c) and (d) in the figure. Therefore, we set $adj = 0.5$ in all our experiments.

III. Performance Evaluation of ASIdE

In this section, we present representative case studies illustrating the effectiveness and the robustness of ASIdE. We first consider stationary workloads where the temporal dependence profile is fixed through the whole experiments and then evaluate the performance of ASIdE under non-stationary workloads with the changing temporal dependence profiles in the end of this section.

We use simulation to evaluate the performability improvement of ASIdE in a network with M servers in series. We assume that there is only one server with temporal dependence in its service process and denote it as Q_{ACF}. The service process at Q_{ACF} is drawn from a MMPP(2) distribution with mean rate $\mu = 1$ job/sec and squared coefficient of variation SCV = 20. We use the autocorrelation function (ACF) to describe and quantify the temporal dependence in the service processes. Let ρ_j be the lag-j autocorrelation coefficient[2]. For the MMPP(2), we maintain the same mean, SCV, and higher moments in all experiments but considering three different autocorrelation profiles:

- *ACF*$_1$: $\rho_1 = 0.47$, ρ_j decays to zero at lag $j = 1400$;
- *ACF*$_2$: $\rho_1 = 0.46$, ρ_j decays to zero at lag $j = 240$;
- *ACF*$_3$: $\rho_1 = 0.45$, ρ_j decays to zero at lag $j = 100$.

Figure 6 shows the ACF for the three profiles. These ACFs are typical and representative of real workloads measurements in storage systems [25], multi-tier architectures [20], and grids [15]. The other $M - 1$ servers, denoted as Q^i_{Exp}, have exponentially distributed service times with mean rate λ_i, $1 \leq i < M$. All these Q^i_{Exp} servers schedule jobs using the FCFS discipline. Q_{ACF} uses different scheduling policies such as FCFS, SJF and ASIdE. We focus on the case where a constant workload of N requests circulates in the network, i.e., the model is a closed queuing network. Simple networks of this type are often used to model real systems, e.g., multi-tier architectures [17], [18], [29]. We remark that service times are independent across servers. For example, there may be a cache thrashing issue on server Q_{ACF} that contributes to burstiness, yet this does not mean that the workload will find the same situation on other servers Q^i_{Exp}.

A. Prediction Accuracy

The key idea in ASIdE is to estimate the expected job service times based on temporal dependence. Here, we first evaluate the performance of the new dependence-based estimator under various autocorrelation profiles. As an ACF profile of an MMPP(2) can usually be characterized by the autocorrelation values at lag 1 (ρ_1) and the autocorrelation decay rate Γ.[3] Here, we evaluate ASIdE's prediction accuracy as a function of ρ_1 and a function of Γ as well. We also define and measure the following two metrics for evaluating prediction accuracy: (1) R_L: the ratio of the number of large jobs that are predicted as large to the total number of large jobs in the system; and (2) R_S: the ratio of the number of small jobs that are predicted as large to the total number of small jobs in the system. The metric of R_L indicates how well ASIdE captures the large ones for delay, while R_S indicates the false positive rate of ASIdE's prediction.

Table I shows that given a long range dependence in service times, i.e., $\Gamma = 0.98$, ASIdE is able to capture the majority of large jobs across all ACF ρ_1 values. When temporal dependence with service times is strong (i.e., $\rho_1 = 0.47$), the false positive rate of ASIdE is quite low. This interprets that ASIdE performs similar to SJF under strong temporal dependence. However, as ρ_1 decreases (i.e., as temporal dependence decreases), the fraction of small jobs that are mistakenly labeled as large increases and R_S becomes closer to R_L, see $\rho_1 = 0.1$. These prediction errors are mainly caused due to the weak correlation among job service times. That is, workloads with weak or no temporal dependence cannot be used to forecast future service times.

[2]The autocorrelation function $\rho_j = \rho_{X_t, X_{t+j}} = \frac{E[(X_t - \mu)(X_{t+j} - \mu)]}{\sigma^2}$ shows the value of the correlation coefficient for different time lags $j > 0$, where μ is the mean and σ^2 is the common variance of $\{X_n\}$.

[3]The autocorrelation decay rate $\Gamma = 2\rho_1/(1 - SCV^{-1})$ indicates the speed of autocorrelation values decaying to zero, in a sense Γ is a measure of short or long range dependence in the process.

TABLE I
PREDICTION ACCURACY OF ASIDE AS A FUNCTION OF ACF VALUES AT LAG 1 ρ_1 (WITH $\Gamma = 0.98$) AND A FUNCTION OF ACF DECAY RATES Γ (WITH $\rho_1 = 0.47$).

	ACF value at lag 1 (ρ_1)			ACF decay rates Γ		
	0.47	0.30	0.10	0.85	0.75	0.65
R_L	0.91	0.85	0.84	0.75	0.62	0.38
R_S	0.03	0.36	0.72	0.30	0.18	0.03

Table I further shows that as the autocorrelation decay rate Γ decreases, less large jobs are captured and thus delayed. Similarly, less small jobs are labeled as large ones and R_S decreases to 0.03 when $\Gamma = 0.65$. This is because when Γ decreases, the ACF values drop quickly to zero at the first few lags. As a result, ASIdE conservatively delays the predicted large jobs and performs closely to FCFS.

B. Performance Improvement

We now simulate a network with two queues: the exponential queue Q^1_{Exp} has mean service rate $\lambda_1 = 2$ job/sec; the autocorrelated queue Q_{ACF} uses the MMPP(2) described above with autocorrelation structure ACF_1, see Figure 6. The model population is set to $N = 500$, the delay limit $D = 1000$.[4] Sensitivity to the most important parameters is explored in the next subsections.

We compare system capacity under ASIdE as measured by the system throughput with the throughput observed when Q_{ACF} uses FCFS or SJF scheduling. Indeed, larger throughput implies that the system can sustain more load, therefore it handles well sudden bursts of requests which improves the overall availability of the system. FCFS performance is used as a baseline for comparison. Our stated goal is to show that ASIdE is competitive to SJF, this implies that the knowledge required by SJF can be inferred effectively from the temporal dependence of the service process.

Table II shows the mean throughput of the different policies and the relative improvement with respect to FCFS. Throughput is measured at an arbitrary point of the network, since for the topology under consideration throughput at steady state must be identical everywhere [6]. The table shows that, although we are not reducing the overall amount of work processed by the system, both with SJF and ASIdE the capacity is significantly better than with FCFS. Noticeably, SJF and ASIdE perform closely, suggesting that the ASIdE approximation of SJF is effective. Also, we remark that ASIdE can work on *any* servers with temporal dependence, which do not need to be the bottleneck.

Further confirmation of this intuition comes from Figure 7(a), which shows the complementary cumulative distribution function (CCDF) of the round trip times, i.e., the probability that the round trip times experienced by individual jobs are greater than the value on the horizontal axis. The plot shows that both SJF and ASIdE enable most of jobs to experience

[4]We conducted experiments with various delay limits, e.g., $D = 50, 100, ..., 1500$ but omitted the results here due to the page limit. We observed that as D increases, ASIdE becomes more aggressive for delaying long jobs and thus achieves more visible performance improvement. However, when D exceeds 1000, such performance improvement diminishes because very few jobs are delayed for more than 1000 times. Thus, we set $D = 1000$ in the remainder of this paper.

TABLE II
MEAN SYSTEM THROUGHPUT (TPUT) AND RELATIVE IMPROVEMENT OVER FCFS FOR A NETWORK WITH $M = 2$ QUEUES, $N = 500$ JOBS, $\lambda_1 = 2$ JOB/SEC AND ACF_1.

	FCFS	ASIdE	SJF
TPUT	0.71 job/sec	0.92 job/sec	1.01 job/sec
% improv.	baseline	29.6%	40.8%

better round trip times compared with FCFS. Indeed, the part of the workload whose execution is delayed at Q_{ACF} receives increased response times, but the number of penalized requests amounts to less than 3% of the total. Observe also that the performance of SJF and ASIdE is close, the only significant difference is that in SJF a small fraction of jobs (less than 0.5%) receives much worse round trip times than in ASIdE. This is attributed to the unavoidable forecasting errors in ASIdE, which may occasionally fail in correctly identifying long jobs, thus resulting in a smaller CCDF tail than SJF.

Other interesting observations arise from Figure 7(b). This figure shows the autocorrelation of the service times at Q_{ACF} under the different scheduling disciplines. Temporal dependence is much less pronounced under SJF and ASIdE, thus suggesting that both policies are able to break the strong temporal locality of the original process.

We further evaluate the performance of short and long jobs separately. Table III presents the mean round trip times, the mean slowdown of short and long jobs, as well as the mean round trip times of *all jobs* under different policies. Figure 8 shows the CCDFs of round trip times for both short and long jobs. Here, round trip time is measured as the sum of response times at all M queues and slowdown is equal to the ratio between job response time and the corresponding job service times (i.e., without any waiting in the queue).

TABLE III
MEAN ROUND TRIP TIME (RTT) OF ALL JOBS, SHORT JOBS (S), AND LONG JOBS (L), AND MEAN SLOWDOWN (SLW) OF SHORT JOBS (S) AND LONG JOBS (L) FOR A NETWORK WITH $M = 2$ QUEUES, $N = 500$ JOBS, $\lambda_1 = 2$ JOB/SEC AND ACF_1.

	RTT	S_RTT	L_RTT	S_SLW	L_SLW
FCFS	701 sec	548 sec	3326 sec	913	201
ASIdE	540 sec	314 sec	4279 sec	523	259
SJF	492 sec	70 sec	7270 sec	116	440

We observe that under both SJF and ASIdE, the overall performance is significantly better than under FCFS, see Table III. Also observe that the distributions of round trip times under ASIdE lie between those under SJF and FIFO for both short and long jobs, see Figure 8. We interpret that ASIdE does not improve the performance of short jobs as much as SJF does because of the inexact information used. However, 99% of small jobs experience better performance under ASIdE than under FCFS. On the other hand, ASIdE does not degrade the performance of long jobs as badly as the SJF does, where about 60% of long jobs experience better performance under ASIdE than under FCFS. By giving higher priority to short jobs, SJF achieves the best performance (e.g., smallest round trip time and slowdown) for short jobs, by compromising the tail in the distribution of round trip times and the slowdown for long jobs. ASIdE achieves moderate fairness (i.e., slowdown)

compared to FCFS and SJF for both short and long jobs.

Finally, we investigate the impact of different values of LT on ASIdE's performance, where LT is used as the threshold to distinguish large service times from small ones. Table IV shows the system throughput of ASIdE as a function of LT, where the parameter k used in Eq. (1) is varied from 0 to 4. We observe that when k (or LT) is large (e.g., $k = 2$ and 4), ASIdE cannot capture most of the expected long jobs and thus has very conservative delaying. As a result, ASIdE obtains similar performance as FCFS. In contrast, when k is small, the performance under ASIdE is significantly improved because large jobs now can be effectively distinguished from small ones and thus be delayed to implicitly approximate SJF. More importantly, the online algorithm shown in Figure 4 successfully self-adjusts LT to a stable value which is exactly equal to $\mu^{-1} + \sigma$.

TABLE IV
MEAN SYSTEM THROUGHPUT (TPUT) OF ASIDE AS A FUNCTION OF LARGE THRESHOLDS $LT = \mu^{-1} + k \cdot \sigma$.

k	0	0.5	1	2	4
LT (sec)	1.00	3.23	5.34	9.92	18.84
TPUT (job/sec)	0.91	0.91	0.92	0.72	0.72

C. Impact of Measurement Delays

The existence of measurement delays is a fundamental challenge in real systems. There always exists some delay between the data being available and the information being recorded. In heavy loaded systems, a significant delay may occur in recording and making available the job completion times. To evaluate the impact of measurement delays on ASIdE's performance, we introduce delay time D_m into the measurement of a job's service time. That is, when a job completes at a server, one cannot immediately obtain that particular job's service time but instead detects the real service time after D_m elapses. To evaluate ASIdE under such delay conditions, we delay the update of conditional probabilities as well as one round of the delaying (i.e., Step 4-a in Figure 3) by D_m.

Figure 9 shows the mean system throughput of ASIdE as a function of measurement delay times for a network with $M = 2$ queues, $N = 500$ jobs, $\lambda_1 = 2$ job/sec and ACF_1. Here, we set the measurement delay time D_m equal to $\mu^{-1}, 2\mu^{-1}, \ldots, 15\mu^{-1}$, where $\mu^{-1} = 1$ sec is the mean service time of Q_{ACF}. One can observe that such measurement delay times can cause the degradation on the quality of ASIdE's predictions as well as ASIdE's performance. However, compared to ASIdE without any measurement delays, i.e., the first bar with $D_m = 0$ in the figure, additional measurement delays do not significantly degrade ASIdE's performance even when $D_m = 15$ seconds, which further verifies the robustness of our new policy in real systems.

D. Sensitivity to Device Relative Speeds

Here, we investigate the robustness of ASIdE's performance to changes in the experimental parameters. Table V summarizes the parameters used in the experiments of Section III-D-Section III-I. Also, in all these experiments we assume the service process stationarity (i.e., the autocorrelation profiles are fixed). Later in the section, non-stationary workloads are also considered.

TABLE V
SUMMARY OF EXPERIMENTAL PARAMETERS IN SENSITIVITY ANALYSIS.

Case	M	N	λ_1	Q_{ACF}	SCV	γ_1
III.C	2	500	1/2/5	ACF_1	20	7
III.D	2	500	2	ACF_1 ACF_2 ACF_3	20	7
III.E	2	500 800 1000	2	ACF_1	20	7
III.F	2/3/4	500	2	ACF_1	20	7
III.G	2	500	2	ACF_1^+	20/40/100	7
III.H	2	500	2	ACF_1^*	20	7/15

We first focus on evaluating networks with varying processing speeds, i.e., we consider the model in Section III-B and vary the service rate at the exponential queue Q^1_{Exp} while keeping fixed the speed at Q_{ACF}. Figure 10 presents the average system throughput for three experiments, labeled $Exp1$, $Exp2$, and $Exp5$, where mean service rate at Q^1_{Exp} is equal to $\lambda = 1, 2$, and 5 job/sec, respectively. As the service rate at Q_{ACF} is $\mu = 1$ job/sec, in $Exp1$ the two queues have identical speed, while in both $Exp2$ and $Exp5$, Q_{ACF} is the system bottleneck and in $Exp5$ the relative speed at Q_{ACF} becomes even slower. The relative capacity improvement with respect to FCFS scheduling is marked above each bar in the figure. The interpretation of the experimental results leads to the following observations.

First, ASIdE improves the system throughput across all experiments and is better for smaller values of λ. The intuition behind this result is that as λ decreases, more jobs are enqueued at the resource Q^1_{Exp}, and then delaying a job produces less overhead because a job put in the tail of Q_{ACF} can still reach the head of the queue quite rapidly. Therefore, the cost of delaying becomes negligible and the network can benefit more of the reordering of jobs.

A second important observation is that as λ increases, the ASIdE performance converges to that of SJF. This suggests that ASIdE forecasting is very accurate since in $Exp5$ almost all population in the network is queuing at Q_{ACF} and SJF sorts nearly perfectly a large population close to N jobs according to their exact size. The fact that ASIdE achieves similar performance indicates that accurate ordering is obtained with forecasting based on temporal dependence.

As a final remark, it is interesting to observe that ASIdE can be more effective than hardware upgrades. For instance, the throughput under ASIdE in $Exp2$ (white bar, $Exp2$) is more than the expected throughout with FCFS in $Exp5$ (black bar, $Exp5$). That is, under temporally dependent workloads, it can be more effective to adopt ASIdE than doubling the hardware speed of Q^1_{Exp}.

We conclude the experiment showing in Figure 11 the CCDF of round trip times. The CCDF tail behavior observed in this subsection persists for $Exp1$, $Exp2$, and $Exp5$, where again ASIdE degrades the performance of only 3% of the total number of requests.

E. Sensitivity to Temporal Dependence

In order to analyze the effect of temporal dependence on policy performance, we conduct experiments with various autocorrelation profiles at Q_{ACF}, but always keep the same mean and variance of the job sizes. We use the three service processes with autocorrelation ACF_1, ACF_2, and ACF_3 shown in Figure 6.

Figure 12 shows the system throughput under FCFS, ASIdE and SJF policies for the same model evaluated in Section III-B but for different autocorrelations. In general, we expect that strong ACF degrades overall system performance more than weak ACF, as it is clearly confirmed by the experimental results. Yet, ASIdE under the stronger ACF improves more than under the weaker ACF. This is because the stronger the ACF, the higher the conditional probabilities for having large-large pairs in the service time series and the delaying is more aggressive. For instance, for ACF_1, we have $P[L|L]_j \geq P[S|L]_j$ for all $j < 69$.

When the service process has the two weaker ACFs, i.e., ACF_2 and ACF_3, the margin for performance improvement of ASIdE and SJF is reduced. In this case, only the conditional probabilities with lags up to $j = 30$ for ACF_2 and up to $j = 14$ for ACF_3 satisfy $P[L|L]_j \geq P[S|L]_j$. This implies that weaker ACFs make ASIdE more conservative in delaying long jobs, but ASIdE still achieves performance very close to the target behavior of SJF.

The plots in Figure 13 present the effect of different temporal dependence on the tail of round trip times under ASIdE. Strong temporal dependence in the service process makes ASIdE delay long jobs more effectively, and thus almost 97% of requests are served up to seven times faster than under the FCFS policy, see Figure 13(a). As temporal dependence becomes weaker in Figure 13(b), the policy delays long jobs less aggressively and a few requests show worse performance. That is, ASIdE becomes less effective, resulting in a longer tail of the round trip times distribution. With low autocorrelation, see Figure 13(c), ASIdE becomes more conservative in delaying jobs, which is reflected by a small fraction of affected jobs. Consistently with the results presented in the previous case studies, SJF gives a long tail in the distribution of round trip times across all experiments. As the strength of ACF decreases, the tail becomes longer.

F. Sensitivity to System Load

Now we investigate the sensitivity of ASIdE to an increased number of requests in the system. This is important because as the network population N increases, the system reaches critical saturation. In order to evaluate how ASIdE improves system availability, we conduct experiments with three different queuing network populations $N = 500$, $N = 800$, and $N = 1000$, while keeping fixed the other parameters as the experiment in Section III-B. The system throughput for these three experiments is illustrated in Figure 14(a). In the experiment with the highest load $N = 1000$, ASIdE improves throughput by 33% compared to FCFS and achieves performance close to the target SJF performance. The improvement is clear also for lower loads, i.e., $N = 500, 800$, but performance gains are maximized under the most congested case for $N = 1000$.

Regarding availability, ASIdE enables the system to sustain higher loads compared to the FCFS policy. For instance, for $N = 800$ and FCFS scheduling, 80% of requests experience round trip times less than 1146 when no delaying of jobs occurs, see the solid curve in plot (b) of Figure 14. However, even for $N = 1000$ requests, the fraction of requests having round trip times less than 1146 becomes 95% with ASIdE (see the dashed curve in plot (b) of Figure 14). That is, ASIdE is able to give a remarkably better performance to most jobs than with FCFS even if the overall population is increased by 200 requests. This makes immediately clear that ASIdE can be very effective in addressing request bursts that threaten system availability.

G. Sensitivity to Network Size

We evaluate the sensitivity of ASIdE to the network size by evaluating throughput improvement for $M = 2, 3, 4$. Except for the autocorrelated queue Q_{ACF}, the remaining $M - 1$ resources are queues with exponential service times. In order to evaluate the different impact of service times that are balanced or unbalanced with respect to the service at Q_{ACF}, we consider the sets of rates shown in Table VI, see the initial part of this section for related notation.

TABLE VI
QUEUE SERVICE RATES IN THE THREE EXPERIMENTS USED TO STUDY ASIDE SENSITIVITY TO DIFFERENT NETWORK SIZES.

M	Q_{ACF}	Q^1_{Exp}	Q^2_{Exp}	Q^3_{Exp}
2	$\mu = 1$	$\lambda_1 = 1$	N/A	N/A
3	$\mu = 1$	$\lambda_1 = 1$	$\lambda_2 = 0.25$	N/A
4	$\mu = 1$	$\lambda_1 = 1$	$\lambda_2 = 0.25$	$\lambda_3 = 1$

Figure 15 shows throughput improvement provided by the three scheduling disciplines. Note that the first experiment is different from the conditions of Table II, since here the two queues are balanced. As the number of queues in the network increases, the relative improvement over the FCFS policy decreases. We interpret this effect by observing that since there are more exponential servers in the network, the temporal dependence of the successive requests at the queues are much weaker than in the experiments considered before. That is, throughout its path, each request is served multiple times by exponential service processes without temporal dependence and therefore the temporal locality effects in the network are reduced. Therefore, the reduced gain in this experiment is rather a consequence of the more limited margin for improvements on these networks rather than a limit of ASIdE. In fact, one can see that SJF improves modestly with respect to the FCFS case.

Also, we observe that consistently with the results presented in the previous cases, ASIdE delays only a small fraction of requests but achieves better performance for most requests. The results are consistent with the properties of ASIdE observed in the previous experiments.

H. Sensitivity to Coefficient of Variation

In order to analyze the effect of the distribution in service times on policy performance, we first investigate the sensitivity of ASIdE to the coefficient of variation (CV) in the service process at Q_{ACF}, which implicitly describes the spread as well as the tail characteristics in the distribution.

Here, we conduct experiments with the exponential queue Q^1_{Exp} having mean service rate $\lambda_1 = 2$ job/sec and the autocorrelated queue Q_{ACF} drawn from MMPP(2) with mean rate $\mu = 1$ job/sec but three different squared coefficient of variations $SCV = 20$, $SCV = 40$, and $SCV = 100$. The model population is set to $N = 500$. The complementary cumulative distribution function (CCDF) of service times at Q_{ACF} is shown in Figure 16(a). Observe that the tail in the service time distribution becomes longer as the variability increases. For instance, $SCV = 100$ has about 4.5 times longer tail than $SCV = 20$.

Figure 16(b)-(d) shows the system throughput under the FCFS, SJF, and ASIdE policies. Different from the previous experiments, the large-job threshold LT in Eq.(1) now is statically set with various $k, 0 \leq k \leq 4$. The goal of exploring different LTs is two-fold: (1) to discuss how the threshold depends on variation; and (2) to confirm how ASIdE effectively computes LT in an online fashion. The results under the case where LT is determined online are also presented in Figure 16 as stripped bars.

The experimental results in Figure 16 demonstrate that there exists a bound for the large-job threshold, beyond which ASIdE does not improve the system performance. This effect can be interpreted by observing that when the large-job threshold is greater than that bound, very few jobs are predicted to be long (i.e., with service times $> LT$) and thus delayed by ASIdE. In addition, the higher the variability, the longer the tail in the job size distribution. As a result, the value of k for that threshold bound increases when SCV changes from 20 to 100.

Observe also in Figure 16 that when LT is below that threshold bound, ASIdE performs nearly the same under various large-job thresholds. This observation indicates that ASIdE is insensitive to LT as long as the value of LT is below some certain value. More importantly, ASIdE always self-adjusts the value of LT to be one among the best choices, see the stripped bars in Figure 16.

I. Sensitivity to Skewness

In this section, we investigate the sensitivity to the skewness in the service process at Q_{ACF}. The skewness in the service process is a measure of the asymmetry of the probability distribution. We observe that compared to FCFS, both ASIdE and SJF significantly improve the system performance when the service process has low skewness. However, as the skewness in the service times increases, the margin of relative performance improvement of both ASIdE and SJF is reduced. This is because the majority of jobs have similar service times and thus few large jobs are delayed by ASIdE and SJF.

J. Non-stationary Workloads

In the previous subsections, we have confirmed that ASIdE performs effectively under a stationary workload. However, it has been shown that because of different transaction types in enterprise applications, the transaction mixes from some production workloads are non-stationary [27]. Therefore, we turn to evaluate ASIdE under non-stationary workloads, further verifying its effectiveness when the workload changes over time.

In particular, we generate a non-stationary workload by changing the temporal dependence profile in its service process of Q_{ACF}. For example, we conduct the same experiments as we did in Section III-B, but instead of having a fixed autocorrelation profile (e.g., only ACF_1 at Q_{ACF}), we here mix different autocorrelation profiles by having the first 30% of jobs with the weakest temporal dependence (i.e., ACF_3), the following 40% of jobs with the strongest temporal dependence (i.e., ACF_1), and finally drawing the remaining jobs from the weaker temporal dependence, i.e., ACF_2. In addition, we vary the service rate at the exponential queue Q^1_{Exp} such that the mean service rate at Q^1_{Exp} is equal to $\lambda = 1, 2$, and 5 job/sec, respectively.

Table VII shows the mean throughput of the three different policies (i.e., FCFS, SJF and ASIdE), as well as the relative improvement with respect to FCFS. We observe that ASIdE successfully captures the changes in temporal dependence profiles and dynamically updates the conditional probabilities and the large-job threshold LT according to the measured performance and temporal dependence strengths. This policy obtains significantly better capacity than FCFS. Noticeably, under the changing workloads, ASIdE still closely approximates the behavior of SJF. These results further demonstrate the effectiveness and robustness of ASIdE under both stationary and non-stationary workloads.

TABLE VII
MEAN SYSTEM THROUGHPUT (TPUT) AND RELATIVE IMPROVEMENT OVER FCFS FOR A NETWORK WITH $M = 2$ QUEUES, $N = 500$ JOBS AND $\lambda_1 = 1, 2$, AND 5 JOB/SEC UNDER NON-STATIONARY WORKLOADS WITH CHANGING AUTOCORRELATION PROFILES.

		FCFS	ASIdE	SJF
Exp1	TPUT	0.64 job/sec	0.79 job/sec	0.92 job/sec
	% improv.	baseline	21.6%	41.7%
Exp2	TPUT	0.80 job/sec	0.96 job/sec	1.00 job/sec
	% improv.	baseline	20.2%	25.8%
Exp5	TPUT	0.91 job/sec	1.00 job/sec	1.01 job/sec
	% improv.	baseline	10.2%	10.6%

Summary of Experiments The extensive experimentation carried out in this section shows that ASIdE can effectively approximate the performance of SJF without the need of additional information about job service times. The sensitivity results on the various autocorrelation profiles have shown that the gains are more pronounced in presence of higher temporally dependent workloads. This suggests that ASIdE is an effective solution to increase the performability of systems processing this type of workloads. Sensitivity analysis to the number of queues in the network and system load show that the gains of ASIdE are visible in a variety of different con-

ditions. By analyzing the effect of the distribution in service times (e.g., the coefficient of variation and the skewness) on policy performance, we further demonstrate that ASIdE *always* selects the appropriate parameters and is highly-competitive to SJF. Finally, the experiments conducted under non-stationary workloads further validate the effectiveness and the robustness of ASIdE.

IV. Case Studies

Now, we validate the effectiveness and robustness of ASIdE with a trace driven simulation from actual measured data. The first real trace was collected by Microsoft Research at an enterprise server for processing project files [21]. Per volume block I/Os (including both reads and writes) were collected below the file system cache over 168 hours in February 2007. Each record describes an I/O request with a timestamp, the disk number, the start logical block number, the number of blocks transferred, and the type (read or write). Here, we consider a multi-tiered enterprise system with limited connections. Our experimental setup assumes two tandem queues: the first one Q_1 represents the front server and the second one Q_2 represents the back-end storage server[5]. The service process of Q_1 is exponentially distributed with mean rate $\lambda = 0.25$ job/msec while the service process of Q_2 is driven by the real I/O trace with mean service rate $\mu = 0.22$ job/msec. We also observe the existence of autocorrelation in the I/O request service times, see Figure 17(a). Therefore, we schedule jobs at Q_1 by FCFS only while at Q_2 by FCFS, ASIdE and SJF policies.

In order to evaluate the performance under different system loads, we increase the number of requests (i.e., the population N) in the system up to 10,000 such that the utilization at Q_2 ranges from 75% to 90%. The system throughput under the three scheduling policies is illustrated in Figure 18(a). Consistently with the previous experiments using synthetic traces, these results validate the effectiveness and the robustness of ASIdE: ASIdE significantly improves the throughput over FCFS, especially when systems are under higher loads. We can interpret this performance improvement the same way as in the previous experiments with synthetic traces (e.g., see Section III-B): (1) ASIdE accurately forecasts the job service times and thus implicitly approximates the behavior of SJF, and (2) ASIdE dramatically decreases the autocorrelation of the service times at Q_{ACF} across all lags. To further investigate the tails of system performance, the CCDFs of the round trip times (the summation of response times at two servers) are plotted in Figure 18(b). Clearly, both SJF and ASIdE allow the majority of requests to experience lower round trip times compared to FCFS.

The second real trace is an HTTP trace collected from the Politecnico di Milano - DEI between September 17th, 2006 and September 24th, 2006. The trace consists of a 2,195,988 rows, each structured according to the extended common logfile format (ECLF) used by the Apache web server. This format contains information about each individual file size transferred to the users of the web site. We assume service times of such files to be proportional to their size; this is useful to illustrate the potential advantage of the ASIdE scheduling if this is implemented in the scheduling algorithm for cache or disk access.

Here, we consider a web server with finite buffers. That is, we conduct experiments with an exponential queue Q_{Exp} representing the arrivals to the HTTP server and an auto-correlated queue Q_{ACF} representing the service process at that server. Q_{Exp} has mean rate of $\lambda = 0.13$ job/sec while the service times in Q_{ACF} are driven by the real HTTP trace with mean rate $\mu = 0.15$ job/sec, squared coefficient of variation $SCV = 25$, and skewness $\gamma = 22$. We remark that the autocorrelation in static object download sizes, shown in Figure 17(b), is not as strong and is rippled across all lags, which makes forecasting job service times more challenging. In order to evaluate the performance under different system loads, we increase the number of requests (i.e., the population N) in the system up to 2000, and thus obtain a utilization level at Q_{ACF} up to 85%.

The system throughput under the three scheduling policies (i.e., FCFS, ASIdE and SJF) is illustrated in Figure 19(a) and the CCDFs of the round trip times and the response times at two queues are plotted in Figure 19(b). Overall, we observe that both ASIdE and SJF improve the system performance (i.e., large throughput) compared to FCFS, which is consistent with the previous experiment using the disk trace. We also observe that the majority of requests obtain the faster round trip times under SJF or ASIdE than under FCFS and the number of penalized requests due to the delay is less than 4% of the total, see Figure 19(b). In general, performance trends are qualitatively similar to those of the synthetic traces.

V. Discussion

A. ASIdE's Optimality

With ASIdE, upon each job completion, the entire queue is scanned to predict the service times of all queuing jobs according to the measured conditional probabilities. We have shown that the effectiveness of the policy relies on the accuracy of the estimates of job service times. If the exact job service time is assumed to be known to the scheduler, then all long jobs to be delayed can be correctly identified, thus ASIdE can perform optimally. Here, we extend ASIdE to an optimal version, called AS_OPT, which marks each queuing job to be long or short based on knowledge of each job's exact service time but still self-adjusts the large-job threshold LT. We remark that the AS_OPT policy provides the bound on the performance of ASIdE. By comparing with AS_OPT, we can investigate the prediction accuracy of ASIdE.

The experiments, which are carried out in Section III for sensitivity analysis on different experimental parameters, are conducted again under the AS_OPT policy. Table VIII gives the results obtained from AS_OPT, as well as those from the other three schedulers. We can observe that FCFS and SJF provide a lower bound and an upper bound, respectively, of ASIdE's performance. Both ASIdE and AS_OPT significantly improve system performability compared to FCFS and

[5]An open system with finite buffers and/or admission control behaves in essence like a system with a closed loop structure [23].

TABLE VIII
MEAN SYSTEM THROUGHPUT (JOB/SEC) UNDER FCFS, ASIDE, AS_OPT AND SJF POLICIES.

(a) Device Relative Speeds

	FCFS	ASIdE	AS_OPT	SJF
Exp1	0.55	0.71	0.76	0.84
Exp2	0.71	0.92	0.97	1.01
Exp5	0.86	0.99	1.00	1.02

(b) Temporal Dependence

	FCFS	ASIdE	AS_OPT	SJF
ACF_1	0.71	0.92	0.97	1.01
ACF_2	0.84	0.98	0.99	0.99
ACF_3	0.91	0.98	0.99	1.00

(c) System Load

	FCFS	ASIdE	AS_OPT	SJF
$N = 500$	0.71	0.92	0.97	1.01
$N = 800$	0.72	0.96	0.99	1.02
$N = 1000$	0.73	0.98	0.99	1.02

(d) Network Size

	FCFS	ASIdE	AS_OPT	SJF
$M = 2$	0.55	0.71	0.76	0.84
$M = 3$	0.38	0.44	0.46	0.48
$M = 4$	0.23	0.24	0.25	0.25

(e) Coefficient of Variation

	FCFS	ASIdE	AS_OPT	SJF
$SCV = 20$	0.80	0.99	1.00	1.01
$SCV = 40$	0.83	0.98	0.98	0.99
$SCV = 100$	0.79	0.99	1.00	1.01

(f) Skewness

	FCFS	ASIdE	AS_OPT	SJF
$\gamma_1 = 7$	0.78	0.99	0.99	1.00
$\gamma_1 = 15$	0.99	0.99	1.00	1.00

are highly-competitive with SJF. More importantly, ASIdE performs closely to the optimal one, i.e., AS_OPT, with a maximum error of 6% in all experiments, suggesting that under a wide variety of system settings ASIdE can achieve good estimates of job service times from the temporal dependence structure of the workload.

B. ASIdE's Classification

In Section III, we showed that ASIdE approximates the performance of SJF by differentiating long and short job service times. In this section, we further investigate the effectiveness of such a binary classification in ASIdE. We first divide all jobs into three classes, i.e., short, medium, and long by drawing job service times at Q_{ACF} from a 3-state MMPP. The sojourn times in each of the three states are exponentially distributed and their means μ_i^{-1} differ by one order of magnitude, i.e., $\mu_i^{-1} = 1, 0.1$ and 0.01 for $i = 1, 2, 3$. This results in high variability with $SCV = 5.4$ and strong temporal dependence with $ACF = 0.4$ at lag 1 in the service process at Q_{ACF}. The service rates at Q_{Exp} and Q_{ACF} are the same and the network population is $N = 200$. We observe that for each job class the relative improvement ratio in terms of round trip times over FCFS is 24.8% for small, 37.1% for medium, and -346.7% for large, respectively. These initial results indicate that under ASIdE the performance improvement mainly comes from delaying the large jobs, resulting in the major contribution to the overall improvement. As the queueing performance is more sensitive to the tail rather than to the body [3], we anticipate that a binary classification which successfully captures the tail is sufficient to achieve good performance.

In order to support the above statement, we refine ASIdE with a ternary classification: upon the completion of a long job, ASIdE scans the entire queue and estimates the size of all the waiting jobs. All expected medium jobs are then delayed by moving them to the end of the queue. Following that, all jobs that are marked long are moved to the end of the queue as well. As a result, ASIdE reshuffles all jobs in the queue based on their anticipated service times: all short jobs are at the head of the queue; all medium jobs are in the middle of the queue; and all long jobs are at the end of the queue.

TABLE IX
MEAN ROUND TRIP TIME (RTT) OF ALL JOBS, SHORT JOBS (S), MEDIUM JOBS, AND LONG JOBS UNDER FCFS AND AS_OPT WITH A BINARY AND A TERNARY CLASSIFICATIONS.

	High CV			
	RRT	S_RRT	M_RRT	L_RRT
FCFS	68.8	31.7	38.1	46.3
AS_OPT - 2	58.6	15.3	16.8	165.5
AS_OPT - 3	58.5	15.2	16.9	165.9
	Low CV			
	RRT	S_RRT	M_RRT	L_RRT
FCFS	57.0	24.4	31.8	34.2
AS_OPT - 2	53.3	14.0	14.4	91.3
AS_OPT - 3	53.3	13.9	15.1	89.9

Table IX shows the results obtained from AS_OPT with a 2-class and a 3-class classification. Here, we focus on the effect of job size classifications by conducting experiments under AS_OPT which provides an upper bound on the performance of ASIdE. We also investigate the performance on various CVs in the service process at Q_{ACF}, i.e., $SCV = 5.4$ for "High CV" case and $SCV = 1.65$ for "Low CV". Table IX shows that AS_OPTs with both classifications significantly improve system performability and more importantly obtain very similar results. This is a good outcome, which verifies that a "short/long" classification is simpler to implement yet sufficient to achieve good system performance.

VI. RELATED WORK

There is a vast literature on scheduling that has been developed over recent years (see [8] and [7] and references therein). Recently, Friedman and Henderson introduce a preemptive scheduling policy for Web servers in [8]. This new policy called Fair Sojourn Protocol (FSP) provides both efficiency and fairness for the sojourn time of the jobs. The SRPT (Shortest-Remaining-Processing-Time) scheduling policy has been presented and analyzed in [11], [12]. However, the SRPT is a preemptive policy while here we focus on the design of a non-preemptive policy. The Priority-based Blind Scheduling (PBS) policy approximates the existing standard blind scheduling policies, e.g., FCFS, PS, and LAS, by tuning

a single parameter [7]. The Generalized Processor Sharing (GPS) policy is studied in the literature [16]. For a two-class GPS system, the admission region is selected for the general Gaussian traffic sources which contain the service processes with both long-range dependence and short-range dependence. The analysis in [10] shows that no single size-independent scheduling policy is optimal for all degrees of correlations in job sizes. To the best of our knowledge, very few general policies consider the structure of temporal locality in scheduling for systems. In [14], [28], [33], a set of scheduling strategies has been proposed from a practical perspective, to deal with correlations in several applications. [4] introduces a non-preemptive application-independent scheduling policy for workloads with correlated job sizes, which uses a class of Hidden Markov Models (HMM) to estimate the sizes of upcoming jobs. Compared to this work, ASIdE can be simpler and more system-oriented. Recently, [24] developed the study on ASIdE, which modeled and evaluated ASIdE under varying workload distributions.

Several papers have investigated the idea of using measured autocorrelation in capacity control policies. In [9] a general framework for measurement-based call admission control is introduced, which uses an approximate Gaussian model of the aggregated traffic to make admission decisions. Similar approaches appear frequently in the networking literature, e.g., in the call admission control in VBR traffic [5], for data communications over CDMA mediums [32], and for general self-similar multiplexed traffic modeled as fractional Brownian motion (fBm) [30]. The above works differ substantially in the scope and approach of the present paper for several reasons. First, network flows can have highly-variable bandwidth requirements that are non-stationary and difficult to model outside heavy traffic or asymptotic regimes; instead service in systems typically shows consistent functional forms which are easier to model and can be exploited effectively to control system load. Another important difference is that network traffic is often modeled as a superposition of flows which share the available bandwidth according to a discriminatory or generalized process-sharing policy; this assumption is instead often unrealistic in systems, e.g., when the scheduling discipline is approximately first-come first-served (FCFS). FCFS scheduling is also found in networks, e.g., in ATM communication, but the service time distributions are here usually deterministic or Erlang, whereas high job size variability in systems is a fundamental factor of congestion.

VII. Conclusions

In this paper, we have proposed ASIdE, a no-knowledge scheduling technique for increasing the performability of systems processing temporally dependent workloads. Temporal locality has been observed in several practical settings, arguing for significant applicability of ASIdE in real systems. Using simulation, we have shown that ASIdE consistently improves performance, as quantified by the system mean throughput and by the distribution of round-trip times experienced by requests under temporal dependent conditions. We have shown that ASIdE is able to approximate effectively the SJF scheduling techniques which is known to provide very effective results in systems, but requiring additional knowledge on job service times that is instead not required by ASIdE. We have also shown the robustness of ASIdE under both stationary and non-stationary workloads and confirmed the accuracy of ASIdE service time forecasting.

Acknowledgments

We would like to thank Feng Yan for his assistance with the Microsoft trace.

References

[1] A. T. Andersen and B. F. Nielsen. On the statistical implications of certain random permutations in markovian arrival processes (MAPs) and second-order self-similar processes. *Perf. Eval.*, 41(2-3):67–82, 2000.

[2] Martin F. Arlitt and Carey L. Williamson. Web server workload characterization: The search for invariants. In *Proc. of ACM SIGMETRICS*, pages 126–137, 1996.

[3] S. C. Borst, O. J. Boxma, and R. Nunez-Queija. Heavy tails: The effect of the service discipline. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Proceedings of TOOLS 2002; LNCS 2324*, pages 1–30. Springer-Verlag, 2002.

[4] Giuliano Casale, Ningfang Mi, and Evgenia Smirni. Cws: a model-driven scheduling policy for correlated workloads. In *Proc. of ACM SIGMETRICS*, volume 38, pages 251–262, 2010.

[5] H.W. Chu, D.H.K. Tsang, and T. Yang. Bandwidth allocation for VBR video traffic in ATM networks. In *Proc. of IEEE ICC*, pages 612–615. IEEE Press, 1995.

[6] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Comp. Surv.*, 10(3):225–261, 1978.

[7] H. Feng, V. Misra, and D. Rubenstein. PBS: a unified priority-based scheduler. In *Proc. of ACM SIGMETRICS*, pages 203–214, New York, NY, USA, 2007. ACM.

[8] Eric J. Friedman and Shane G. Henderson. Fairness and efficiency in web server protocols. In *Proc. of ACM SIGMETRICS*, pages 229–237, New York, NY, USA, 2003. ACM.

[9] M. Grossglauser and D. N. C. Tse. A framework for robust measurement-based admission control. *IEEE/ACM T. Networking*, 7(3):293–309, 1999.

[10] Varun Gupta, Michelle Burroughs, and Mor Harchol-Balter. Analysis of scheduling policies under correlated job sizes. *Performance Evaluation*, pages 336–345, 2010.

[11] M. Harchol-Balter, N. Bansal, B. Schroeder, and M. Agrawal. Srpt scheduling for web servers. In *7th International Workshop, Job Scheduling Strategies for Parallel Processing*, pages 11–20, 2001.

[12] M. Harchol-Balter, M.E. Crovella, and S. S. Park. The case for SRPT scheduling in web servers. Technical Report MIT-LCS-TR-767, MIT Laboratory for Computer Science, October 1998.

[13] A. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Science/Engineering/Math, 1999.

[14] A. A. Lazar, G. Pacifici, and D. E. Pendarakis. Modeling video sources for real-time scheduling. In *Multimedia Sys.*, pages 835–839, 1993.

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

[16] P. Lieshout, M. Mandjes, and S. Borst. GPS scheduling: selection of optimal weights and comparison with strict priorities. In *Proc. of ACM SIGMETRICS/Performance*, pages 75–86, 2006.

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

[18] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *ACM/IFIP/USENIX International Middleware*, pages 265–286, 2008.

[19] N. Mi, G. Casale, A. Riska, Q. Zhang, and E. Smirni. Autocorrelation-driven load control in distributed systems. In *Proceedings of MASCOTS*, Sept. 2009.

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

[21] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write off-loading: Practical power management for enterprise storage. *Trans. Storage*, 4:10:1–10:23, November 2008.

[22] N.Mi, G.Casale, and E.Smirni. Scheduling for performance and availability in systems with temporal dependent workloads. In *In Proceedings of IEEE DSN*, 2008.
[23] R. O. Onvural and H. G. Perros. Equivalencies between open and closed queueing networks with finite buffers. *Performance Evaluation*, 9:263–269, 1989.
[24] I. A. Rai and M. Okopa. Modeling and evaluation of swap scheduling policy under varying job size distributions. In *Proc. of ICN)*, pages 115–120, 2011.
[25] A. Riska and E. Riedel. Long-range dependence at the disk drive level. In *Proc. of QEST*, pages 41–50. IEEE Press, 2006.
[26] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Oper. res.*, 16:687–690, 1968.
[27] Christopher Stewart, Terence Kelly, and Alex Zhang. Exploiting nonstationarity for performance prediction. In *Proc. of ACM SIGOPS/EuroSys*, pages 31–44, 2007.
[28] W. Uchida, T. Hara, and S. Nishio. Scheduling correlated broadcast data considering access frequencies with temporal variations. In *Proc. of IEEE NCA*, pages 89–97, 2003.
[29] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi. An analytical model for multi-tier internet services and its applications. In *Proc. of SIGMETRICS*, pages 291–302. ACM, 2005.
[30] J.L. Wang and A. Erramilli. A connection admission control algorithm for self-similar traffic. In *Proc. of IEEE GLOBECOM*, pages 1623–1628. IEEE Press, 1999.
[31] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4:419–420, 1962.
[32] J. Zhang, M. Hu, and N. Shroff. Bursty data over CDMA: MAI selfsimilarity, rate control, and admission control. In *Proc. of IEEE INFOCOM*, 2002.
[33] X. Zhong and C. Z. Xu. Energy-aware modeling and scheduling of real-time tasks for dynamic voltage scaling. In *Proc. of IEEE RTSS*, pages 366–375, 2005.

Ningfang Mi Ningfang Mi is an Assistant Professor at Northeastern University, Department of Electrical and Computer Engineering, Boston, MA 02115 (ningfang@ece.neu.edu). She received her Ph.D. degree in Computer Science from the College of William and Mary, VA in 2009; her doctoral dissertation was on dependence-driven techniques in system design. She received her M.S. in Computer Science from the University of Texas at Dallas, TX in 2004 and her B.S. in Computer Science from Nanjing University, China, in 2000. Her research area mainly focuses on capacity planning, resource management, performance evaluation, simulation, virtualization, and cloud computing. She is a member of ACM, IEEE, and IEEE Computer Society.

Giuliano Casale Giuliano Casale received the M.Eng. and Ph.D. degrees in computer engineering from Politecnico di Milano, Italy, in 2002 and 2006 respectively. He joined in 2010 the Department of Computing at Imperial College London where he holds an Imperial College Junior Research Fellowship. His research interests include performance modeling, workload characterization, and resource management. He is co-author of the Java Modelling Tools performance evaluation suite (http://jmt.sf.net). Prior to joining Imperial College, he was a full-time researcher at SAP Research UK in 2009 and a postdoctoral research associate at the College of William and Mary, Virginia, in 2007 and 2008. In Fall 2004 he was a visiting scholar at UCLA. He serves as program co-chair for QEST 2012 and for ACM SIGMETRICS/Performance 2012. He is a member of ACM, IEEE, and IEEE Computer Society. From 2012 he serves as secretary/treasurer of the IEEE STC on Sustainable Computing.

Evgenia Smirni Evgenia Smirni received the diploma degree in computer engineering and informatics from the University of Patras, Greece, in 1987, and the PhD degree in computer science from Vanderbilt University in 1995. Currently she is a Professor of Computer Science at the College of William and Mary, Williams- burg, Virginia. Her research interests include stochastic models, matrix analytic methods, resource allocation policies, data centers, Internet and multi-tiered systems, storage systems, workload characterization, and modeling of distributed systems and applications. She has served as a program cochair of QEST 2005, of the ACM SIGMETRICS/Performance 2006, and of HotMetrics10. She has also served as a general cochair of QEST10 and NSMC'10. Since June 2011 she is serving in the ACM Sigmetrics Board of Directors. She is a member of the ACM, the IEEE, and the Technical Chamber of Greece.

1. input:
- **a.** constant population of the system N;
- **b.** number of classes for service times C;
- **c.** service time thresholds $\{T_0 = 0, T_1, ..., T_C = \infty\}$;
- **d.** large service time threshold $LT \leftarrow T_{C-1}$;

2. **for each monitoring window** T_W
- **a.** initialize:
  - **I.** num. of jobs $N\_L \leftarrow 0$ for service times $> LT$;
  - **II.** num. of $C_k|L$ pairs $N^j\_C_kL \leftarrow 0$, $1 \leq k \leq C$ and $1 \leq j < N$;
- **b** **for each job completion**
  - **b.1.** if its service time $> LT$, then
  - **I.** $N\_L \leftarrow N\_L + 1$;
  - **II.** for each j-th job arrived before and completed,
    - if its service time $S_j \in [T_{k-1}, T_k)$ then $N^j\_C_kL \leftarrow N^j\_C_kL + 1$;
  - **III.** for each j-th job arrived after and completed,
    - if its service time $S_j > LT$ then $N^j\_C_CL \leftarrow N^j\_C_CL + 1$;
  - **b.2** if its service time $\in [T_{k-1}, T_k)$, then
  - **I.** for each j-th job arrived after and completed
    - if its service time $S_j > LT$ then $N^j\_C_kL \leftarrow N^j\_C_kL + 1$;
- **c.** **at the end of the window** T_W
  update conditional probabilities at lag j, $1 \leq j < N$
  - **I.** $P[C_k|L]_j \leftarrow N^j\_C_kL/N\_L, 1 \leq k \leq C$;

Fig. 1. Description of how to calculate condition probabilities.

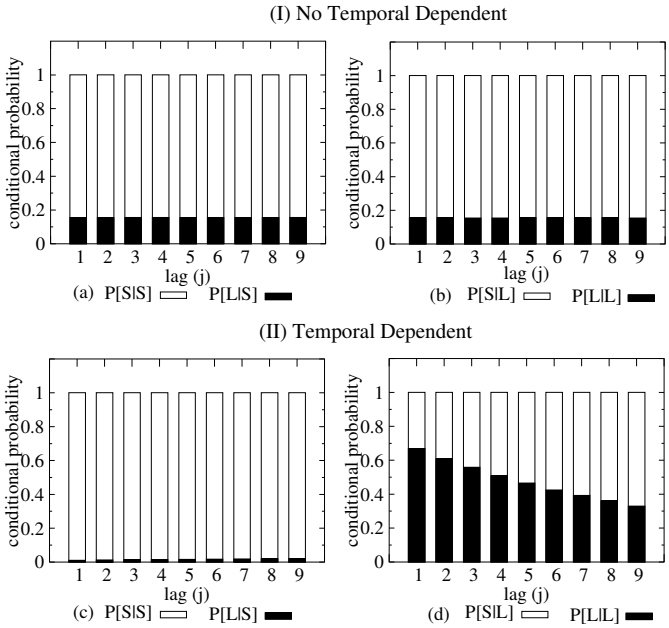

Fig. 2. Conditional probabilities as a function of lags j. Plots (a) and (b) give results for a temporally independent sequence. Plots (c) and (d) give results for a temporally dependent sequence.

1. input:
- **a.** maximum allowable delay limit D;
- **b.** constant population of the system N;
- **c.** num. of classes for service times $C \leftarrow 2$;
- **d.** arrival index $i \leftarrow 0$;
- **f.** large threshold $LT \leftarrow \mu^{-1} + k \cdot \sigma$;

2. **for each job arriving at queue**
- **a.** $i \leftarrow i + 1$;
- **b.** set that job's arrival index to i;
- **c.** initialize that job's predicted result as UnChecked;
- **d.** initialize that job's num. of delays $d \leftarrow 0$;

3. **for each monitoring window** T_W
- **a.** recalculate conditional probabilities $P[C_k|L]_j$, $1 \leq k \leq C$ and $1 \leq j < N$, as shown in Fig. 1;

4. **for each large job ($> LT$) completion at queue**
- **a.** trigger one round of the delaying;
  - **I.** initialize $j \leftarrow 1$;
  - **II.** if predicted result of the j-th job is not UnChecked, then keep using its predicted result;
  - **III.** if predicted result of the j-th job is UnChecked, then predict the size of the j-th job;
    - – calculate the lag apart the two jobs as j-th job's arrival index - completed job's arrival index;
    - – if $P[C_{k*}|L]_j = \mathbf{max}\{P[C_1|L]_j, ..., P[C_C|L]_j\}$ then set that job's predicted result as class C_{k*};
  - **IV.** $j \leftarrow j + 1$;
  - **V.** if not reaching the end of the queue, then go to **step 4-II.**; else for each class C_k job with num. of delays $d \leq D$, for $1 < k \leq C$
    - – delay it to the end of the queue;
    - – set $d \leftarrow d + 1$;

Fig. 3. Description of ASIdE.

1. initialize: $\mu \leftarrow 0$, $\sigma \leftarrow 0$, $k \leftarrow 1$, and $adj \leftarrow 0.5$;
2. set $LT \leftarrow \mu^{-1} + k \cdot \sigma$;
3. for each request in a monitoring window T_W do
  a. upon each job completion at the autocorrelated server
    I. compute observed conditional probabilities: $P[L|L]_j$, for $1 \leq j < N$, as shown in Fig. 1;
    II. update μ^{-1} and σ by Welford's algorithm [31];
    III. update the mean queue length Q;
  b. at the end of the window T_W
    I. if $P[L|L]_{\lfloor Q/2 \rfloor} \geq P[S|L]_{\lfloor Q/2 \rfloor}$,
    then $k \leftarrow k + adj$;
    else if $P[L|L]_{\lfloor Q/10 \rfloor} < P[S|L]_{\lfloor Q/10 \rfloor}$,
    then $k \leftarrow k - adj$;
    II. set maximum and minimum large thresholds:
    $LT\_max \leftarrow 90^{th}$ percentile of observed service times;
    $LT\_min \leftarrow 50^{th}$ percentile of observed service times;
    III. recalculate $LT \leftarrow \mu^{-1} + k \cdot \sigma$;
    IV. if $LT > LT_{max}$, then $LT \leftarrow LT_{max}$;
    V. if $LT < LT_{min}$, then $LT \leftarrow LT_{min}$;

Fig. 4. Description of how to self-adjust LT.

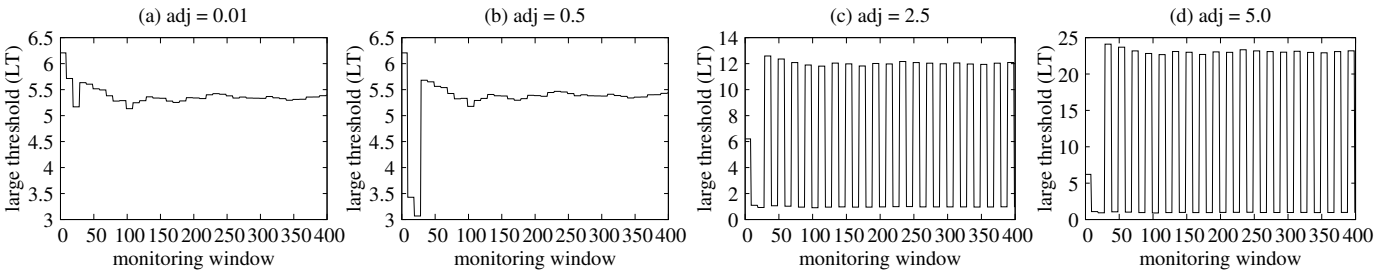

Fig. 5. Illustrating the adjustment of large threshold LT given different values of adj in Figure 4.

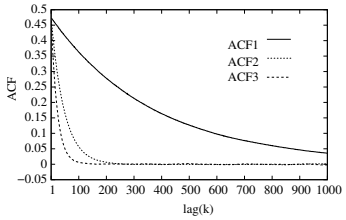

Fig. 6. The ACF of the service process that generates the autocorrelated flows in the system, where the service times are drawn from MMPP(2) with ACF_1, ACF_2 and ACF_3, respectively.

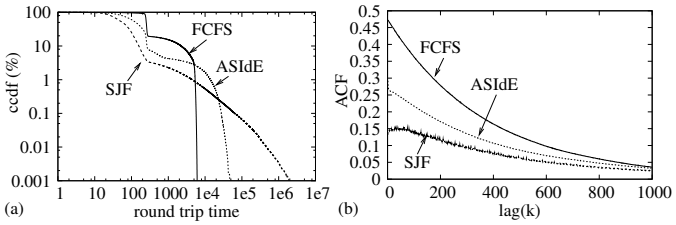

Fig. 7. Comparisons of ASIdE, SJF and FCFS: (a) CCDF of round trip times, (b) autocorrelation (ACF) of service times at Q_{ACF}.

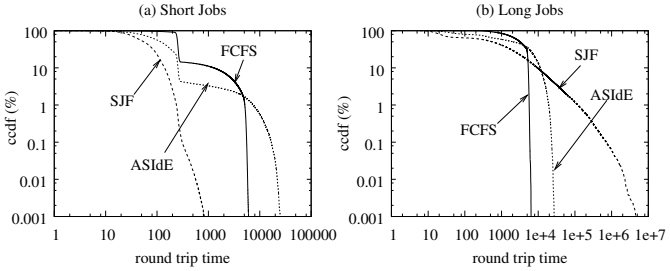

Fig. 8. CCDFs of round trip times of short and long jobs for a network with $M = 2$ queues, $N = 500$ jobs, $\lambda_1 = 2$ job/sec and ACF_1.

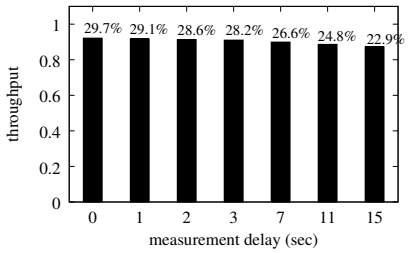

Fig. 9. Mean system throughput of ASIdE as a function of measurement delay times (in seconds) for a network with $M = 2$ queues, $N = 500$ jobs, $\lambda_1 = 2$ job/sec and ACF_1. The relative improvements over FCFS are plotted on each bar.

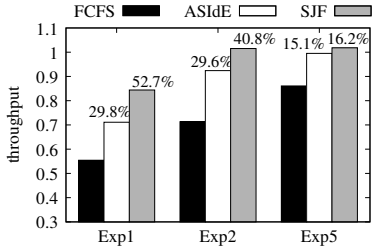

Fig. 10. Sensitivity to service process ratio in a network with $M = 2$, $N = 500$, and ACF_1.

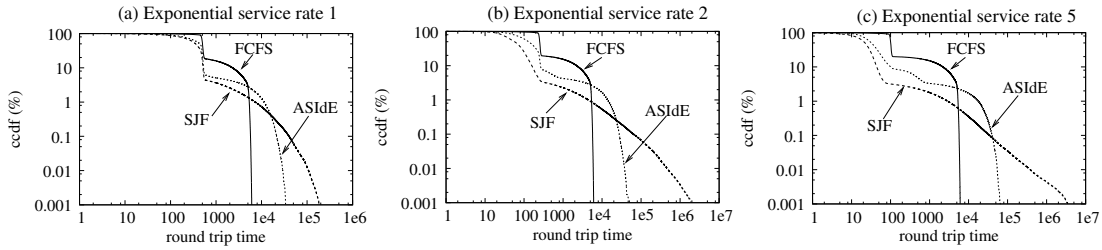

Fig. 11. Illustrating the CCDF of round trip times in a network with $M = 2$, $N = 500$, and ACF_1. The service rate λ_1 of the exponential queue is equal to (a) 1, (b) 2, and (c) 5 job/sec.

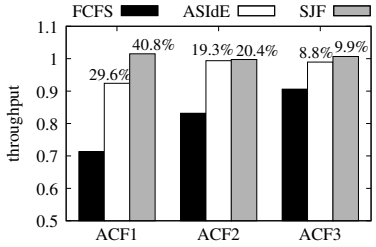

Fig. 12. Sensitivity to temporal dependence in a network with $M = 2$, $N = 500$, and $\lambda_1 = 2$ job/sec.

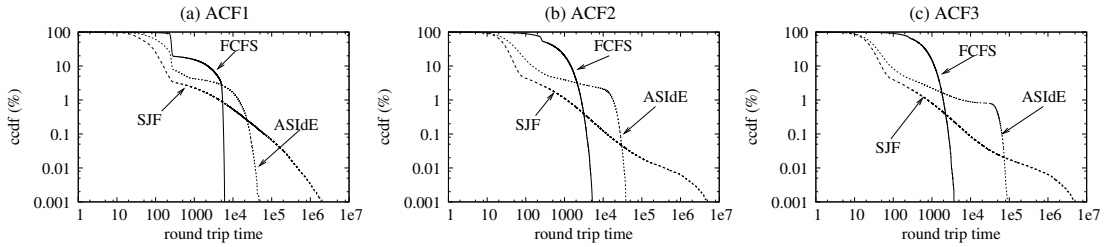

Fig. 13. Illustrating the CCDF of round trip times in a network with $M = 2$, $N = 500$, and $\lambda_1 = 2$ job/sec. The service process of Q_{ACF} has temporal dependence (a) ACF_1, (b) ACF_2, and (c) ACF_3.

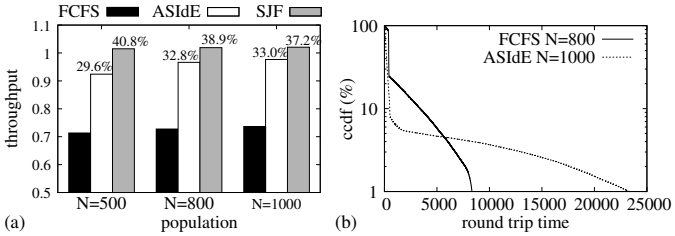

Fig. 14. Sensitivity to network population in the system with $M = 2$, $\lambda_1 = 2$ job/sec, and ACF_1. Illustrating (a) average system throughput, where the relative improvement over the FCFS policy is indicated on each bar; and (b) CCDFs of round trip times, where the solid curve shows the results in the experiment with $N = 800$ under FCFS and the dashed curve presents the results in the experiment with $N = 1000$ under ASIdE.

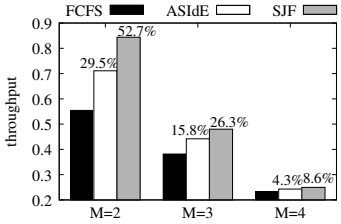

Fig. 15. Sensitivity to network size in a network with $N = 500$ and ACF_1.

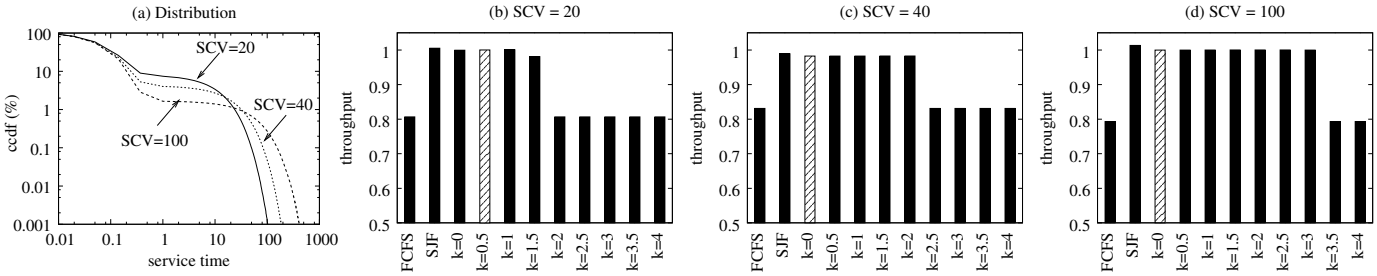

Fig. 16. Sensitivity to coefficient of variation (CV) of service times at Q_{ACF} in a network with $M = 2$, $N = 500$, and $\lambda_1 = 2$ job/sec. Illustrating CCDF of service times at Q_{ACF} and performance results (e.g., system throughput), where the service times at Q_{ACF} are drawn from MMPP(2) with $\mu = 1$ job/sec and $SCV = 20$, 40, and 100, and the static large-job threshold for ASIdE is set to $\mu^{-1} + k \cdot \sigma$, for $0 \leq k \leq 4$. The results for ASIdE with online calculated LT are marked by striped bars.

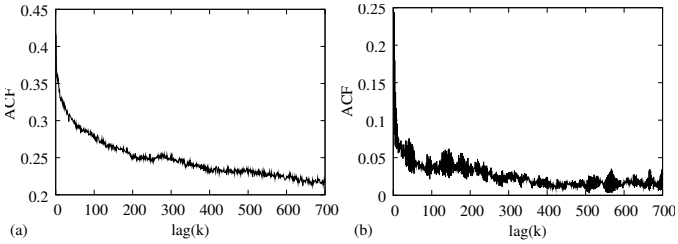

Fig. 17. Illustrating ACFs of (a) block I/O service times of the storage server and (b) static object download sizes (i.e., the service process) of the HTTP server.

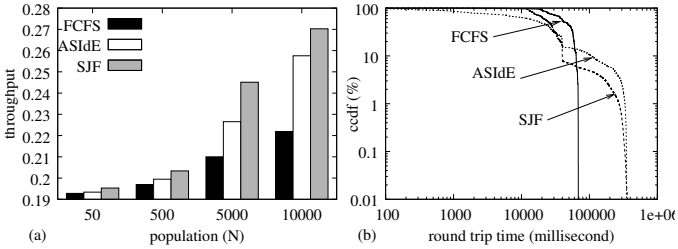

Fig. 18. Block I/O Trace: (a) overall system throughput as a function of network population (N) under FCFS, ASIdE and SJF, and (b) CCDFs (tails) of round trip times in the system with $N = 10000$.

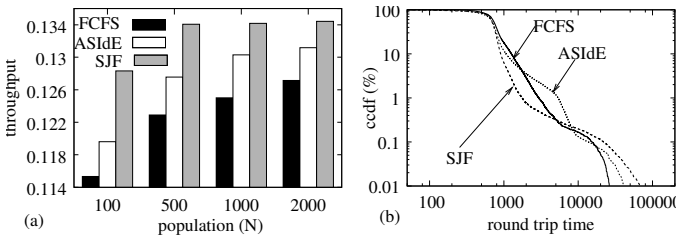

Fig. 19. HTTP: (a) overall system throughput as a function of network population (N) under FCFS, ASIdE and SJF, and (b) CCDFs of round trip times and response times in a network with $N = 100$.