# Scheduling for Performance and Availability in Systems with Temporal Dependent Workloads[*]

Ningfang Mi, Giuliano Casale, and Evgenia Smirni
Computer Science Department, College of William and Mary
{ningfang,casale,smirni}@cs.wm.edu

## Abstract

*Temporal locality in workloads creates conditions in which a server, in order to remain available, should quickly process bursts of requests with large service requirements. In this paper, we show how to counteract the resulting peak congestions and maintain high availability by delaying selected requests that contribute to the temporal locality.*

*We propose and evaluate SWAP, a measurement-based scheduling policy that approximates the shortest job first (SJF) scheduling* without *requiring any knowledge of job service times. We show that good service time estimates can be obtained from the temporal dependence structure of the workload and allow to closely approximate the behavior of SJF. Experimental results indicate that SWAP significantly improves system performability. In particular, we show that system capacity under SWAP is largely increased compared to first-come first-served (FCFS) scheduling and is highly-competitive with SJF, but without requiring a priori information of job service times.*

## 1 Introduction

Temporal dependence in workloads processed by multi-tier architectures, disk drives, and grid services, significantly reduces performance and availability by creating peak congestions that can make service unavailable [9, 12, 14]. With temporal locality, requests with large service requirements frequently appear clustered together, reducing system throughput for a period that is usually much longer than the duration of the arrival burst. Thus, system availability is strongly dependent on the response given to these workload peaks and efficient schemes to address bursts become fundamental for performance and availability.

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., MMPP, fractional Brownian motion, $M/G/\infty$) [18], and to measurement-based load-control schemes for network availability under rapidly changing flows [7]. Unfortunately, because of systematic violations of the underlying assumptions, these schemes cannot be easily applied to systems. Deterministic or Erlang service time distributions in ATM networks are replaced in systems by highly-variable service demands [2, 14]; similarly, the usual assumption that a channel multiplexes a sufficiently large number of traffic flows to enable Gaussian approximations [7] is not fit for systems where restrictive constraints on the maximum concurrency level exist, e.g., limits on the maximum number of simultaneous HTTP sessions or DB locking conditions.

In this paper, we address the lack of effective measurement-based schemes to maintain performance and availability in systems with temporal dependent workloads. We focus on the difficult case where processing the entire workload is mandatory and where work reduction techniques such as request drop cannot be applied. Our main contribution is to show that significant performance gains and high system availability can be obtained by delaying selected requests that contribute to temporal locality. We observe that request delaying results in significant throughput improvement throughout the network, thus allowing delay-based scheduling to increase the amount of request that a server can process at a given time and therefore avoiding harmful congestion conditions. We interpret the significant performance improvement as an outcome of the autocorrelation reduction in the service process of the resource where requests are delayed. Since the temporal dependence propagates through the entire network [12], by decreasing the autocorrelation at the resource with delay-based scheduling, we also reduce the autocorrelation in the arrival process of the other servers in the network, which results in generalized throughput improvement. We observe that larger throughput lets the system sustain more customers, therefore improving the overall availability. We also remark that this simple approach can be more effective than hardware

[*] This work was supported by the National Science Foundation under grants ITR-0428330 and CNS-0720699.

upgrades, i.e., doubling the speed of a server is not as effective as implementing a delay-based scheduling technique at the server with temporal dependent workload.

Delay-based scheduling is investigated throughout the paper by defining a new scheduling policy, called SWAP. SWAP is a fully measurement-based policy that classifies (i.e., "predicts") requests as short or long based on the temporal dependence of the workload service process. That is, we leverage on the structure of temporal locality to forecast the size of upcoming requests and define self-adjusting criteria to discriminate (i.e., delay) requests that the algorithm deems as large, to be delayed. Experimental results indicate that SWAP can increase throughput up to $30\% - 40\%$ under temporal dependent workloads, without service rejection while maintaining the fraction of delayed requests low. These results show that SWAP works comparably to Shortest Job First (SJF), despite the fact that it does not require any a priori knowledge of future workload.

Sensitivity analyses with respect to device relative speeds, to different degrees of temporal dependence, to system load, and to network size show that SWAP is effective and robust in many different environments. Furthermore, we also show that in all cases, the performance under SWAP is very close to that of the SJF policy, thus suggesting the effectiveness of the workload prediction used by SWAP.

This paper is organized as follows. In Section 2, we present the SWAP scheduling algorithm. In Section 3 we use simulation to validate the effectiveness and robustness of SWAP. In Section 4 we give an overview of the previous work in networking on scheduling and availability control under temporal dependent workloads. Finally, in Section 5, we draw conclusions and outline future work.

## 2 Delay-Based Scheduling Policy: SWAP

In this section we introduce SWAP, a new delay-based scheduling policy that improves performance and availability in systems with temporal dependent workloads. The basic idea behind SWAP can be summarized as follows. Consider a system processing jobs with a first-come first-served (FCFS) scheduling policy. Assume that the *exact job size* information is available to the scheduler. If we want to maximize performance given that the future instants of new job arrivals are unknown, then the optimal scheduling is shortest job first (SJF) as it is well-know from classic scheduling theory [15]. That is, if the resource has $K$ enqueued jobs having ordered service times $S_k$, $1 \leq k \leq K$, being $S_1$ the service time required by the job at the head of the queue, the total completion time $CT$ under the FCFS discipline is

$$CT = KS_1 + (K-1)S_2 + \ldots + S_K,$$

which represents the time interval from the moment that the first job arrives to the moment that the last job leaves the service center, and is immediately minimized if $S_k \leq S_{k+1}$, i.e., when short jobs are served first.

Outside the above assumptions, SJF is not in general optimal, but yet provides significant gains with respect to simpler scheduling policies such as FCFS. We therefore investigate how the performance of SJF could be approximated with an online policy that does not require a priori knowledge. That is, the well-know problem of SJF is that it requires information on the job service times, which in practice may not be available. The basic idea behind SWAP is to use the measured serial correlation of the service times to estimate this missing information. Once these reliable estimates of the job service times are available, we delay large jobs up to a fixed number of times by putting them at the tail of the queue. In such a way, long jobs are more likely to be served after most short jobs have been completed. Estimated-short jobs are not delayed by SWAP.

Summarizing, the basic ideas of SWAP are as follows:

1. approximate the behavior of the SJF scheduling discipline by proper use of job delaying;
2. estimate the expected service times of the jobs waiting in queue from the process temporal dependence, as modeled by the correlation between successive service time values.

We stress that SWAP does *not* assume any a priori knowledge of the length 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 the remaining jobs is based only on the past history of the system. We also stress that we provide mechanisms to avoid job starvation. In the following subsections we detail the SWAP policy and its implementation.

### 2.1 Forecasting Job Service Times

The effectiveness of the new 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 SWAP performs optimally. We first present SWAP service time forecasting methodology which is specifically tailored to temporal dependent workloads.

**Exploiting Service Time Variability**

Our service time forecasting relies on two system aspects: service time variability and temporal dependence of workloads. Concerning the former, we leverage on the fact that service time distributions found in systems are typically characterized by high variance [2, 14] and therefore the discrimination between small and large service times can be performed effectively and used to improve performability. In particular, SWAP uses a large-job threshold ($LT$)

$$LT = \mu^{-1}(1 + k \cdot CV), \qquad (1)$$

1-4244-2398-9/08/$20.00 ©2008 IEEE

where $\mu^{-1}$ is the mean service time at the resource, $CV$ is the coefficient of variation of service times (i.e., the ratio of the standard deviation to the mean), and $k \geq 1$ is a constant determined online. If a job service time is greater than $LT$, then SWAP regards the job as "long" (also referred throughout the paper as "large"). Otherwise SWAP classifies it as "short". Note that the policy can successfully measure the parameters for computing $LT$ in an online fashion, i.e., the mean $\mu^{-1}$ and the coefficient of variation $CV$ of the service times are continuously updated in SWAP using Welford's one-pass algorithm [8].

**Exploiting Temporal Dependence**

Given a classification into large and short jobs, the next step to effective forecasting is to exploit the structure of temporal dependence in order to "guess" if a job in the queue is long or short. This is the critical information needed to approximate the behavior of SJF scheduling. We assume that the scheduler is able to measure correctly the service times of jobs completed by the server; this is readily available in most systems. Let $T$ be the time instant in which a forecasting decision is needed, which in SWAP always corresponds to the departure instant of a *long* job departing from the queue. Also assume that during the period $[T - T_W, T]$, the system has completed $n$ jobs with service times $S_1, S_2, \ldots, S_n$. $T_W$, $0 \leq T_W \leq T$, is an update window monitoring past history. Given the sequence $\{S_i\}$, $1 \leq i \leq n$, our forecasting is based on the estimates of the conditional probabilities

$$\begin{aligned} P[L|L]_j =& P[S_{t+j} \geq LT | S_t \geq LT], \\ P[S|L]_j =& P[S_{t+j} < LT | S_t \geq LT] = 1 - P[L|L]_j, \end{aligned}$$

which are computed using the samples $S_t \in \{S_i\}$ for $t = 1, \ldots, n - j$. Here $j$ is called the lag of the conditional probability and denotes the distance between the service completions considered in the conditional probabilities. Given that the last completed job is long, $P[L|L]_j$ measures the fraction of times that the $j$-th job that had arrived after it is also long; similarly, $P[S|L]_j$ estimates how many times the lag-$j$ arrival is short. Using these estimates, we forecast that the lag-$j$ arrival after the last completed job is going to receive large service time if the following condition holds

$$P[L|L]_j \geq P[S|L]_j, \tag{2}$$

i.e., there is higher probability that the $j$-th arrival is going to be long than short. SWAP is triggered only when the last finished job is long; therefore, since we focus on closed systems only, i.e., systems with constant population $N$, we *only* make use of the conditional probabilities $P[L|L]_j$, for $1 \leq j < N$.

```
1. initialize:
   a. maximum allowable delay limit $D$;
   b. arrival index $i \leftarrow 0$;
   c. large threshold $LT \leftarrow \mu^{-1}(1 + k \cdot CV)$;
2. upon 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. upon each job completion at queue
   a. measure conditional probabilities $P[L|L]_j$, $1 \leq j < N$;
   b. if its service time is greater than $LT$,
      then 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 $P[L|L]_{lag} \geq P[S|L]_{lag}$,
             then set that job's predicted result as large;
             else set that job's predicted result as small;
      IV.  $j \leftarrow j + 1$;
      V.   if reaching the end of the queue,
           then for each large job with num. of delays $d \leq D$
           delay it to the end of the queue and set $d \leftarrow d + 1$;
           else, go to step 3-c-II;
   c. else, go to step 3;
```

**Figure 1.** Description of SWAP.

## 2.2 The Delaying Algorithm: SWAP

We now describe SWAP in detail. For presentation simplicity, we assume here that the large threshold $LT$ that is fundamental for forecasting is given; in the next subsection, we present how SWAP self-adjusts $LT$ on-the-fly, i.e., no a priori knowledge of $LT$ is required and SWAP becomes truly autonomic.

Upon the completion of a long job, the entire queue is scanned and the size of the $j$-th queued job is predicted by using the conditional probabilities introduced in the previous subsection. If the $j$-th job is estimated as large, SWAP marks it as such. All jobs that are marked long are delayed by moving them at the end of the queue. After all jobs in the queue have been examined and long jobs have been delayed, SWAP admits for service the first job in the queue. Delaying is not triggered again before completion of another long job.

Jobs are "reshuffled" 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 in stochastic processes that can be modeled using Markovian methods.

SWAP does not re-forecast the length of a job whose service time has been already forecasted to be long. This is done by recording an absolute arrival index $A_i$ for each job. That is, once a job has been marked as long it remains as such for all the duration of its stay in the queue and is never forecasted as short in successive activations of SWAP; the same property holds also for short jobs. We apply the conditional probabilities on the sequence of jobs in the queue obtained by ordering the jobs according to the arrival indexes only.

To avoid starvation of long jobs, we introduce the *delay limit* $D$, i.e., 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. Figure 1 gives the pseudocode of SWAP and summarizes the above discussion.

### 2.3 Self-Adjusting the Threshold $LT$

Now, we discuss how SWAP adjusts the threshold $LT$ for large values, aiming at controlling the strength of delaying 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, the performance improvement is then negligible. Conversely, when $LT$ becomes too small, more jobs (even short ones) are delayed and therefore throughput is reduced. As a result of this, performance may be improved very little. Therefore, the choice of an appropriate large threshold $LT$ is critical for the effectiveness of SWAP.

As observed in Section 2.1, the computation of $LT$ is a function of the updating window $T_W$ used by SWAP. We express $T_W$ as the maximal time period in which the system has completed exactly $W$ requests; in the experiments presented here, we set $W = 100,000$. The algorithm in Figure 2 describes how the threshold $LT$ is dynamically adjusted every $W$ requests. At the end of a period of length $T_W$, we update $LT$ while keeping as upper and lower bounds for its value the 90th and the 50th percentiles of the observed service times in $T_W$. 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 $LT = \mu^{-1}(1 + k \cdot CV)$ with step $adj$ according to the following scheme. Let $QT$ be the current queue-length at the server with SWAP scheduling. We evaluate $P[L|L]_j$ for the large lag $j = \lfloor QT/2 \rfloor$ and if $P[L|L]_j \geq P[S|L]_j$, then SWAP is assumed to be too aggressive, since it may delay at the next round[1] a number of jobs up to $\lfloor QT/2 \rfloor$. In this case we set $k = k + adj$, which reduces the number of jobs identified to be long. A similar procedure is performed for the case $j = \lfloor QT/10 \rfloor$, where if $P[L|L]_j \geq P[S|L]_j$, we conventionally assume that SWAP is too conservative; in this case we set $k = k - adj$ which increases the number of jobs estimated as large. Throughout experiments we have always observed that the $LT$ online algorithm does not show instability problems and always provide effective choices of $LT$ which lead to consistent performance gains as discussed in the next section.

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

**Figure 2.** Description of how to self-adjust $LT$.

## 3 Performance Evaluation of SWAP

In this section, we present representative case studies illustrating the effectiveness and the robustness of SWAP. For all simulations, we generate the service time traces with 10 million samples. Simulations stop only after all the service times have been used. Throughout all experiments, we stress that SWAP never changes the statistical distribution of the service times and the ratio between long and short jobs. Instead, SWAP only reorders the service times while keeping the distribution intact. We stress that although we

[1] Here we implicitly assume that the conditional probabilities $P[L|L]$ are decreasing in $j$ which indeed is the typical case for workloads where large service times are a minority compared to the small service times.

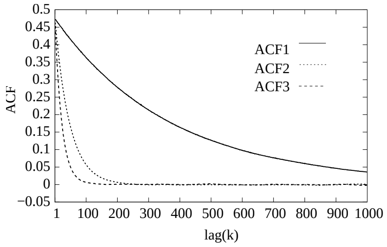

**Figure 3.** The ACF of the service process that generates the autocorrelated flows in the system, where the service times are drawn from MMPP(2) with *ACF*$_1$, *ACF*$_2$ and *ACF*$_3$, respectively.

simulate a closed system with jobs that cycle in the two servers, every time when a job arrives in a service station we cast a random number to generate its service time. Thus, in our simulations we never re-cycle short jobs and never change the distribution of service times.

We use simulation to evaluate the performability improvement of SWAP in a network with $M$ first-come-first-served (FCFS) servers in series. We assume that there is only one server with temporal dependence in its service process and denote that queue as $Q_{ACF}$. Throughout experiments, the service process at $Q_{ACF}$ is always a two-state Markov-Modulated Poisson Process (MMPP(2)) [13] with identical distribution having mean rate $\mu = 1$ and squared coefficient of variation $CV^2$ = 20. Let $\rho_j$ be the lag-$j$ autocorrelation coefficient. For the MMPP(2) we consider three different autocorrelation profiles:

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

Figure 3 shows the ACF for the three profiles. These ACFs are typical and representative of real workloads measurements in storage systems [14], multi-tier architectures [12], and grids [9]. The other $M - 1$ queues, denoted as $Q^i_{Exp}$, have exponentially distributed service times with mean rate $\lambda_i$, $1 \leq i < M$. We focus on the case where a constant workload of $N$ requests circulates in the network, i.e., the model is a closed queueing network. Simple networks of this type are often used to model real systems, e.g., multi-tier architectures [11, 16].

### 3.1 Performance Improvement

We first simulate a network with two queues: the exponential queue $Q^1_{Exp}$ has mean service rate $\lambda_1 = 2$; the autocorrelated queue $Q_{ACF}$ uses the MMPP(2) described above with autocorrelation structure *ACF*$_1$. The model population is set to $N = 500$, the delay limit is $D = 100$. Sensitivity to the most important experiment parameters is explored in the next subsections.

We compare system capacity under SWAP as measured by the system throughput with the throughputs observed when $Q_{ACF}$ uses FCFS or SJF scheduling. Indeed, larger throughput means that the system can sustain more load and it is protected from the degradation of sudden bursts of requests, which improves the overall availability of the system. FCFS performance is used for a baseline in comparison. We recall that our stated goal is to show that SWAP is competitive to SJF which would show that the knowledge required by SJF can be inferred effectively from the temporal dependence of workloads.

| | FCFS | SWAP | SJF |
|---|---|---|---|
| TPUT | 0.71 job/sec | 0.92 job/sec | 1.01 job/sec |
| % improv. | baseline | 29.6% | 40.8% |

**Table 1.** Mean system throughput (TPUT) and relative improvement over FCFS for a network with $M = 2$ queues, $N = 500$ jobs, $\lambda_1 = 2$ and autocorrelation profile *ACF*$_1$.

| | FCFS | SWAP | SJF |
|---|---|---|---|
| Overall RTT | 701 sec | 540 sec | 473 sec |
| short RTT | 548 sec | 314 sec | 70 sec |
| long RTT | 3326 sec | 4279 sec | 7270 sec |

**Table 2.** Mean round trip time (RTT) of all jobs, short jobs, and long jobs, for a network with $M = 2$ queues, $N = 500$ jobs, $\lambda_1 = 2$ and autocorrelation profile *ACF*$_1$.

Table 1 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 [4]. The table shows that, although we are not reducing the overall amount of work processed by the system, both with SJF and SWAP the capacity is significantly better than with FCFS. Noticeably, SJF and SWAP perform closely, thus suggesting that the SWAP approximation of SJF is very effective.

Table 2 further presents the mean round trip times (RTTs) of short and long jobs. The mean round trip times of *all jobs* for different policies are presented as well. Round trip time is measured as the sum of response times at all $M$ queues. Table 2 shows that under both SJF and SWAP policies, the overall performance is significantly better than under FCFS. Because of the inexact information used, SWAP does not improve the performance of short jobs as much as SJF does. On the other hand, SWAP does not degrade the performance of long jobs as worse as the SJF does. By giving the higher priority to short jobs, SJF achieves the long

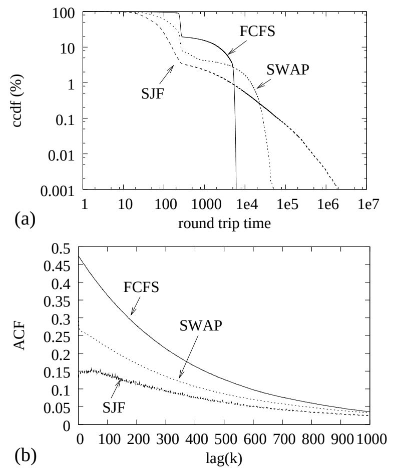

**Figure 4.** Comparative evaluation of SWAP, SJF and FCFS: (a) CCDF of RTTs, (b) autocorrelation (ACF) of service times at $Q_{ACF}$.

tail in the distribution of round trip times for long jobs.

Further confirmation of this intuition comes from Figure 4(a), which shows the complementary cumulative distribution function (CCDF) of the round trip times (RTTs), 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 the largest part of job experiences the lowest RTTs when the scheduling is SJF or SWAP. 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 SWAP is extremely close. The only significant difference is that in SJF a small fraction of jobs (less than $0.5\%$) receives much worse RTTs than in SWAP. We attribute such difference to the unavoidable forecasting errors in SWAP, which may occasionally fail in identifying jobs as long also if their actual service requirement is large, thus resulting in a smaller CCDF tail than SJF.

Other interesting observations arise from Figure 4(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 SWAP, thus suggesting that both techniques are able to break the strong temporal locality of the original process.

## 3.2 Sensitivity to Device Relative Speeds

From now on, we investigate the robustness of SWAP performance to changes in the experimental parameters. We first focus on evaluating networks with varying processing speeds, i.e., we consider the model in Section 3.1 and vary the service rate at the exponential queue $Q^1_{Exp}$ while keeping fixed the speed at $Q_{ACF}$. Figure 5 presents the average system throughput for three experiments, labeled $Exp1$, $Exp2$, and $Exp5$, where we set $\lambda = 1, 2$, and $5$, respectively. For ease of comparison, and only for this experiment, we have rescaled in experiments the mean service rate at $Q_{ACF}$ to $\mu = 2$. Thus, in $Exp1$ the slowest bottleneck queue is $Q^1_{Exp}$, in $Exp2$ the two queues have identical speed, while in $Exp5$ the bottleneck is $Q_{ACF}$. 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.

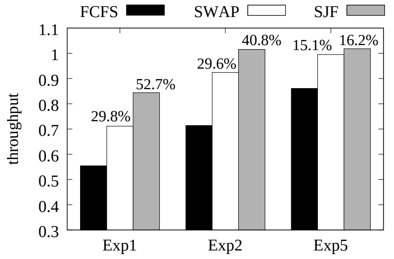

**Figure 5.** Sensitivity to service process ratio in a network with $M = 2$, $N = 500$, and $ACF_1$. The numbers on the top of bars are the improvements with respect to FCFS.

First, SWAP improves the system throughput across all experiments and is better for smaller values of $\lambda$. The intuition behind this result is that if $Q^1_{Exp}$ is the bottleneck, then delaying a job produces less overhead, i.e., a job put in the tail of $Q_{ACF}$ can yet reach the head of the queue quite rapidly since most of the network population is enqueued at the other resource $Q^1_{Exp}$. In this way, the cost of delaying becomes negligible and the network can benefit more of the reordering of jobs sizes.

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

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

We conclude the experiment showing in Figure 6 the CCDF of RTTs for the previous experiments. The CCDF tail behavior observed in the previous subsection persists

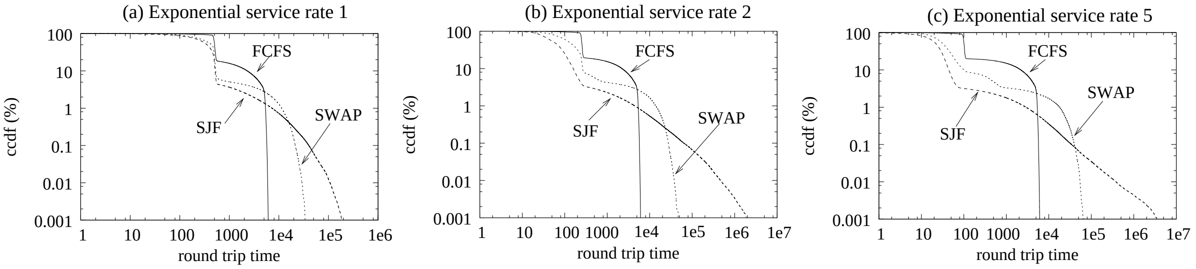

**Figure 6.** Illustrating the CCDF of RTTs in a network with $M = 2$, $N = 500$, and $ACF_1$. The service rate $\lambda_1$ of the exponential queue is equal to (a) 1, (b) 2, and (c) 5.

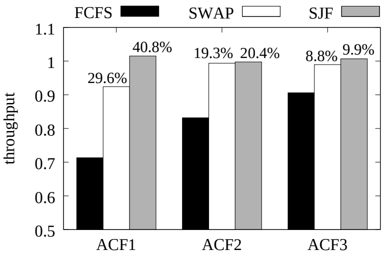

**Figure 7.** Sensitivity to temporal dependence in a network with $M = 2$, $N = 500$, and $\lambda_1 = 2$, where the relative improvement over the FCFS policy is indicated on each bar.

for $Exp1$, $Exp2$, and $Exp5$, where again SWAP degrades the performance of only $3\%$ of the total number of requests.

### 3.3 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 keeping the same mean and $CV$ of the job sizes. We use the three service processes with autocorrelation $ACF_1$, $ACF_2$, and $ACF_3$ shown in Figure 3.

Figure 7 shows the system throughput under FCFS, SWAP and SJF policies for the same model evaluated in Section 3.1 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, SWAP 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 SWAP and SJF is much 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 SWAP more conservative in delaying long jobs, but SWAP still achieves performance very close to the target behavior of SJF.

The plots in Figure 8 present the effect of different temporal dependence on the tail RTTs under SWAP. Strong temporal dependence in the service process makes SWAP to 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 8(a). As temporal dependence becomes weaker in Figure 8(b), the policy delays long jobs less aggressively and a few requests show worse performance. That is, SWAP becomes less effective, resulting in a longer tail of the RTTs distribution. With low autocorrelation, see Figure 8(c), SWAP 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 RTTs across all experiments and as the strength of ACF decreases, the tail becomes longer.

### 3.4 Case 4: Sensitivity to System Load

Now we investigate the sensitivity of SWAP to an increased number of requests in the system. This is extremely important to understand the performability benefit of the technique as the system reaches critical congestion. In order to evaluate how SWAP improves system availability, we conduct experiments with three different network populations $N = 500$, $N = 800$, and $N = 1000$, while keeping fixed the other parameters as the experiment in Section 3.1. The system throughput for these three experiments is illustrated in Figure 9 and the CCDFs of the RTTs experienced by individual requests are plotted in Figure 10. In the experiment with the highest load $N = 1000$, SWAP 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 maximal under the most congested case $N = 1000$.

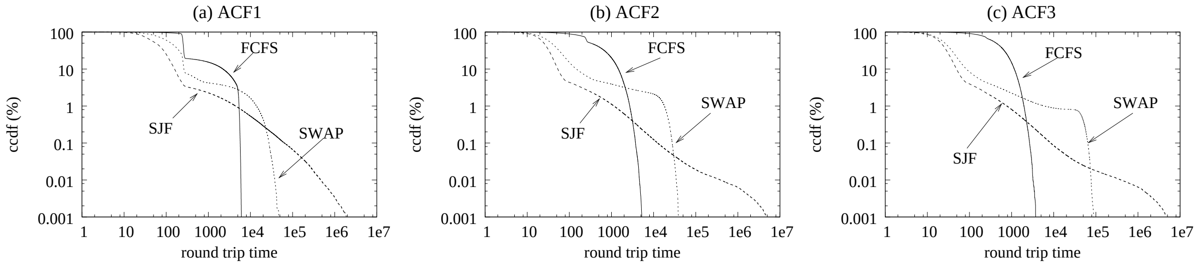

**Figure 8.** Illustrating the CCDF of RTT in a network with $M = 2$, $N = 500$, and $\lambda_1 = 2$. The service process of $Q_{ACF}$ has temporal dependence (a) *ACF*$_1$, (b) *ACF*$_2$, and (c) *ACF*$_3$.

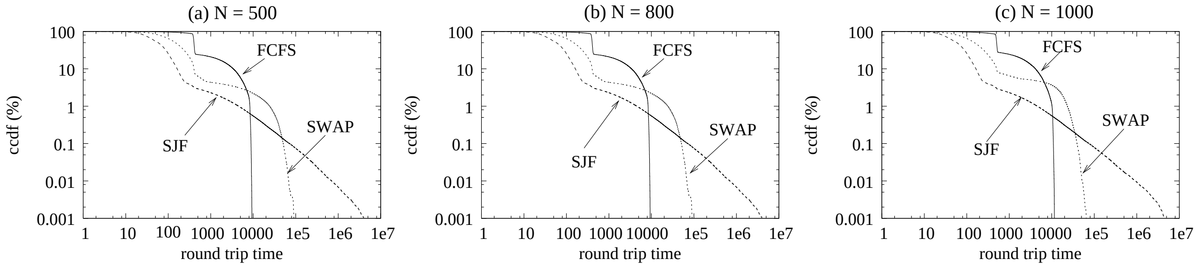

**Figure 10.** Illustrating the CCDF of RTT in a network with $M = 2$, $\lambda_1 = 2$, *ACF*$_1$. The network population is (a) $N = 500$, (b) $N = 800$, and (c) $N = 1000$.

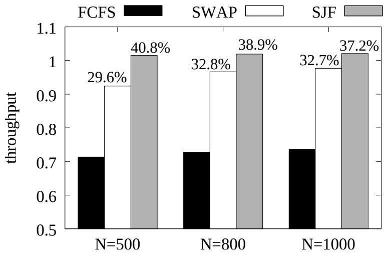

**Figure 9.** Sensitivity to network population in the system with $M = 2$, $\lambda_1 = 2$, and *ACF*$_1$, where the relative improvement over the FCFS policy is indicated on each bar.

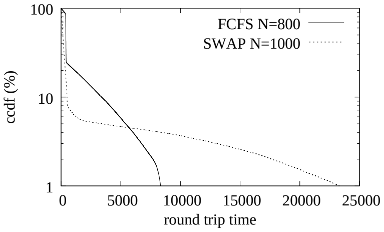

**Figure 11.** Illustrating the CCDF of RTT in a network with $M = 2$, $\lambda_1 = 2$, and *ACF*$_1$. The solid curve shows the results in the experiment with $N = 800$ under the FCFS policy and the dashed curve presents the results in the experiment with $N = 1000$ under SWAP.

Regarding availability, SWAP enables the system to sustain higher loads compared to the FCFS policy. For instance, for $N = 800$ and FCFS scheduling, $80\%$ of requests experience RTTs less than $1146$ when no delaying of jobs occurs, see the solid curve in Figure 11. However, even for $N = 1000$ requests, the fraction of requests having round trip times less than 1146 becomes $95\%$ with SWAP (see the dashed curve in Figure 11). That is, SWAP 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 SWAP can be very effective in addressing request bursts that threaten system availability.

### 3.5 Sensitivity to Network Size

We evaluate the sensitivity of SWAP 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 3, see the initial part of this section for related notation.

Figure 12 shows throughput improvement provided by the three scheduling disciplines. Note that the first experi-

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

**Table 3.** Queue service rates in the three experiments used to study SWAP sensitivity to different network sizes.

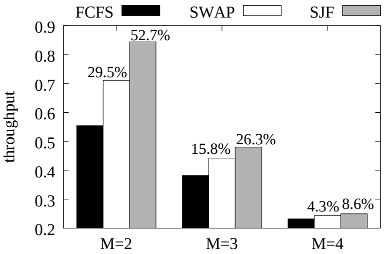

**Figure 12.** Sensitivity to network size in a network with $N = 500$ and $ACF_1$, where the relative improvement over the FCFS policy is indicated on each bar.

ment is different from the conditions of Table 1, 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 network rather than a limit of SWAP. In fact, we see that also SJF improves modestly with respect to the FCFS case. Preliminary experiments indicate that if SJF or SWAP are deployed at multiple queues with temporal dependent workloads, then overall system performance and availability benefit significantly. We leave the detailed investigation of these properties of SWAP for future work.

We complete the analysis in this subsection with Figure 13 that plots the CCDF of job RTTs for the three experiments. Consistently with the results presented in the previous cases, SWAP delays only 2-3% of requests but achieves better performance for most requests. The results are consistent with the properties of SWAP observed in the previous experiments, and the results are almost indistinguishable across the three experiments.

### 3.6 Summary of Experiments

The extensive experimentation carried out in this section has revealed that SWAP 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 proved that the gains are more pronounced in presence of higher temporal dependent workloads. This suggests that SWAP is an effective solution to increase performability in systems processing this type of workloads. Sensitivity analyses to the number of queues in the network and system load show that the gains of SWAP are visible in a variety of different conditions.

## 4 Related Work

A large literatures on scheduling policies (disciplines) have been developed over the years (see [6] and [5] and references therein). Recently, Friedman and Henderson introduce a preemptive scheduling policy for Web servers in [6]. This new policy called Fair Sojourn Protocol (FSP) provides both efficiency and fairness for the sojourn time of the jobs. 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 [5]. The Generalized Processor Sharing (GPS) policy is studied in the literature [10]. 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. However, to our best knowledge, no existing policy considers the structure of temporal locality in scheduling for systems.

Several papers have investigated the idea of using measured autocorrelation in capacity control policies. In [7] a general framework for measurement-based call admission control is introduced. Admission decisions are taken by means of an approximate Gaussian model of the aggregated traffic which is parameterized by the measured mean, variance and correlation of the superposed flows. Similar approaches appear frequently in the networking literature, e.g., in the call admission control in VBR traffic [3], for data communications over CDMA mediums [18], and for general self-similar multiplexed traffic modeled as fractional Brownian motion (fBm) [17]. However, these 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 approx-

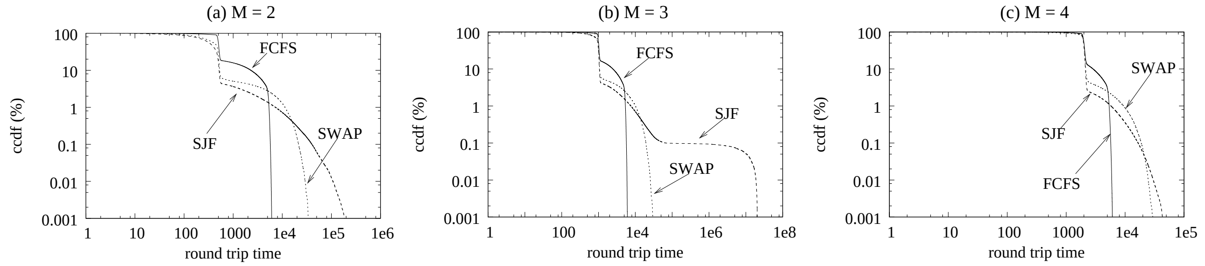

**Figure 13.** Illustrating the CCDF of round trip time in a network with $N = 500$ and $ACF_1$. The number of queues in the network is (a) $M = 2$, (b) $M = 3$, and (c) $M = 4$.

imately 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.

## 5 Conclusions

In this paper, we have proposed SWAP, a no-knowledge scheduling technique for increasing the performability of systems processing temporal dependent workloads. Temporal locality has been observed in several practical settings, arguing for significant applicability of SWAP in real systems. Using simulation, we have shown that SWAP 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 SWAP 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 SWAP.

Future work includes the extension of SWAP for scheduling multiple classes of jobs, the analytical modeling of the maximum achievable performance gains under temporal dependent workloads and the additional validation of SWAP in complex distributed networks, e.g., in systems that are best modeled by several queues with temporal dependent service.

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

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

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

[5] H. Feng, V. Misra, and D. Rubenstein. PBS: a unified priority-based scheduler. In *Proc. of ACM SIGMETRICS '07*, pages 203–214, New York, NY, USA, 2007. ACM.

[6] E. J. Friedman and S. G. Henderson. Fairness and efficiency in web server protocols. In *Proc. of ACM SIGMETRICS '03*, pages 229–237, New York, NY, USA, 2003. ACM.

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

[8] S. K. P. L. H Leemis. *Discrete-Event Simulation: A First Course*. Prentice Hall, 2005.

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

[10] P. Lieshout, M. Mandjes, and S. Borst. GPS scheduling: selection of optimal weights and comparison with strict priorities. In *Proc. of ACM SIGMETRICS '06/Performance '06*, pages 75–86, New York, NY, USA, 2006. ACM.

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

[12] 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.

[13] M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, New York, 1989.

[14] A. Riska and E. Riedel. Long-range dependence at the disk drive level. In *Proc. of 3rd Conf. on Quantitative Evaluation of Systems (QEST)*, pages 41–50. IEEE Press, 2006.

[15] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Oper. res.*, 16:687–690, 1968.

[16] 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.

[17] J. Wang and A. Erramilli. A connection admission control algorithm for self-similar traffic. In *Proc. of IEEE GLOBECOM*, pages 1623–1628. IEEE Press, 1999.

[18] J. Zhang, M. Hu, and N. Shroff. Bursty data over CDMA: MAI selfsimilarity, rate control, and admission control. In *Proc. of IEEE INFOCOM*, 2002.