# Towards More Effective Utilization of Computer Systems

Niklas Carlsson
Linköping University
Linköping, Sweden
niklas.carlsson@liu.se

Martin Arlitt
HP Labs
Palo Alto, USA
martin.arlitt@hp.com

## ABSTRACT

Globally, vast infrastructures of Information Technology (IT) equipment are deployed. Much of this infrastructure is under utilized to ensure acceptable response times. This results in less than ideal use of the capital investment used to purchase the IT equipment. To improve the *sustainability* of IT, we focus on increasing the effective utilization of computer systems. Our results show that computer systems running *delay-sensitive* (e.g., Web) workloads can be more effectively utilized while still maintaining adequate (e.g., mean or upper percentile) response times. In particular, these computer systems can simultaneously support *delay-tolerant* workloads, to increase the value of work done by a computer system over time.

## Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems

## General Terms

Performance, Design

## Keywords

Analytical modeling, effective utilization, percentile analysis

## 1. INTRODUCTION

The tremendous popularity of the World Wide Web has resulted in the deployment of vast infrastructures to support interactive (i.e., delay-sensitive) workloads. These infrastructures typically have average utilizations between 10 and 50% [1], owing to the diurnal variations in usage and the need to keep response times low. The need to support such workloads cost effectively and the desire to minimize the environmental footprint of Information Technology (IT) has motivated many researchers to develop techniques for reducing the energy consumption of servers and data centers. While beneficial, such techniques address only a small part of the Total Cost of Ownership (TCO) of a data center, where the capital cost of the IT, power, cooling and related infrastructure can account for up to 80% of the TCO [2]. Thus, to improve the Return on Investment (ROI) for the data center owner, the effective utilization of the infrastructure must be improved [3]. This is similar to what is done in other industries, such as manufacturing and refining, which typically operate plants 24 hours a day. A related challenge is finding workloads that are more valuable than their cost of operation, so that the increased utilization is economically feasible. Towards this challenge, we identify general properties such workloads must exhibit.

Our work leverages the fact that some jobs[1] are more tolerant to delay than others. As long as progress can be made on *delay-tolerant* jobs while completing *delay-sensitive* jobs quickly, the effective utilization of the IT infrastructure can be improved. By showing this approach is theoretically viable, we can help mitigate the conservative use of computer systems in practice.

We find that it is possible to improve the effective use of computer systems, up to 100% of capacity in some cases, by combining delay-tolerant and delay-sensitive workloads on the same system. An important observation from our results is that by minimizing the variation in the size of tasks in the delay-tolerant workload, the mean response times in delay-sensitive workloads can remain adequately low even as the computer system's overall utilization increases. We also examine the effects on the tail of the response times, as values such as the 99th percentile often are important in practice. Our analysis confirms our intuition that the tail typically is even less sensitive to the addition of delay-tolerant jobs.

An underlying message of the paper is that there is great value in understanding the workloads running on a system. For example, if delay-tolerant workloads with high job-size variability can be separated from those with low variability, jobs with low variability can effectively share resources with the delay-sensitive jobs while any delay-tolerant workloads with high variability may be better processed separately. As the demands of the delay-sensitive workloads changes with time, so may the best way to operate the system. Overall, we show that there is great value in careful workload scheduling and server-resource management. Our results also suggest that most of the benefits of resource management over longer time periods can be achieved using a relatively simple pol-


*ICPE'11,* March 14–16, 2011, Karlsruhe, Germany.


[1]We use the term *job* generically, to represent any task, transaction, or process that consumes IT resources.

icy that greedily maximizes the resource utilization for the current conditions.

The remainder of the paper is organized as follows. Section 2 provides background information and related work. Section 3 introduces our model framework. Section 4 describes the steady state analysis of our model. Section 5 analyzes our model under a time-varying workload. Section 6 discusses possible extensions to our work. Section 7 summarizes our contributions and lists future research directions.

# 2. BACKGROUND AND RELATED WORK

Over the past several decades, Information Technology (IT) has becoming increasingly entwined in everyday life. This trend is expected to continue, as more uses for IT and IT services are identified. This trend, coupled with recent global financial uncertainties and concern over climate change, motivate increased scrutiny of the economic and environmental *sustainability* of IT.

In the systems research community, *efficiency* is a common topic. Over the past decade, connection scheduling has been extensively researched to improve the efficiency of Web servers; [4] includes a thorough summary. A concern with adopting new scheduling mechanisms is their *fairness*. Schroeder and Harchol-Balter demonstrate that even under overload, the SRPT mechanism does not discriminate against large jobs while making the Web server more efficient [4]. Such works are relevant in that they investigate the issue of *starvation* [5]. Our work is similar in that we aim to keep *delay-sensitive* workloads (e.g., Web transactions) from excessive delays. We differ in that we also consider *delay-tolerant* workloads, which have less restrictive definitions of starvation.

*Consolidation* is another technique for making more efficient use of computer systems. *Virtualization* is a technology that facilitates consolidation, by supporting multiple virtual machines on each physical machine. Kusic *et al.* [6] and Zhu *et al.* [7] both supplement virtualization with control techniques to improve the effective utilization of physical IT resources automatically, while maintaining adequate application performance. Chen *et al.* [8] extend this, by coupling the management of power, cooling and IT resources together to further reduce energy use.

From a more theoretical perspective, Wierman *et al.* [9] examine how to optimally speed scale modern CPUs to balance mean response time and mean energy consumption under processor sharing scheduling. Andrew *et al.* [10] extend this work, and find that the evaluated designs can only achieve two of fairness, optimality and robustness. Several works combine analytical and experimental techniques. Gandhi *et al.* [11] look at how to achieve the maximum performance from a fixed power budget. Meisner *et al.* [12] propose a method for eliminating the power consumed by a server during idle periods. While such methods are important for reducing operating costs, they do not enable a data center owner to take full advantage of their capital investment.

Pruhs *et al.* [13] consider the "bi-criteria optimization problem" of minimizing average responses time and energy use. They point out that most prior works formulate the problem as a scheduling problem, but that in generalized settings, most jobs do not have natural deadlines. Abdelzaher *et al.* [14] attempt to address this issue by applying real-time scheduling theory to aperiodic tasks. While this approach has the potential to improve the utilization of computer systems, the lack of natural deadlines still limits this in practice.

Pruhs *et al.* also indicate that mean response time is "by far the most commonly used QoS measure in the computer systems literature" [13]. Thus, a gap exists between theory and practice, as the upper tail of the response time distribution is of greater interest to operators [15]. Unfortunately, there is a lack of research that considers percentiles rather than moments. In 1967, Gaver [16] pointed out that models that used moments to model computer systems typically oversimplified the variability of randomness of such systems. Lazowska [17] proposed the use of percentiles in modeling, to enable more accurate results. However, since then the use of percentiles garnered little attention in the modeling community. An exception to this is the work of Bodik *et al.* [18], which model the performance of the tail, motivated by [15].

Our analysis builds upon theory for M/G/1 queuing systems with vacation periods [19, 20]. For comprehensive surveys of vacation models, we refer the interested reader to works by Doshi [19, 20] or Takagi [21], for example. In our analysis, vacation periods are used to serve delay-tolerant jobs and/or to model idle periods. Other works have considered discriminatory processor sharing (DPS) and the performance of server policies for systems with jobs of different priorities [22, 23, 24]. Perhaps the most similar work where vacation models have been applied is the work by Nunez-Queija *et al.* [25]. They consider the blocking probabilities and file transfer delays over a network link (e.g., ATM or IP) that serves both (delay-sensitive) streaming traffic and (delay-tolerant) elastic traffic. Similar to us, they find that there sometimes is value for the workloads to share resources, rather than using separate resources for the two workloads. However, focusing on a different domain they are concerned with a quite different service model and metrics. Further, they do not consider the tail-behavior of the delay-sensitive workload.

There are tools for executing delay-tolerant workloads on computer systems that may be running delay-sensitive workloads. For example, Condor was developed to more effectively utilize workstations [26], while Condor-G can provide high throughput computing in a grid environment [27]. Our theoretical results may motivate increased use of such tools in practice.

Lastly, Karidis *et al.* suggest using larger, more expensive servers to improve data center ROI, as such servers can be more effectively used than smaller, inexpensive servers [3]. Our work extends theirs by considering the upper tail of response times, and by combining delay-sensitive and delay-tolerant workloads, to increase the effective utilization of computer systems over time. We also consider smaller, less expensive servers.

# 3. MODEL FRAMEWORK

## 3.1 System Abstraction

We are interested in maximizing the utilization of computer system resources. We assume a system for which both *delay-sensitive* and *delay-tolerant* workloads are of interest to the operator. For example, a system may support a delay-sensitive Web application as its *prioritized workload*, and a more delay-tolerant batch-style analysis application as its

*background workload*. We assume the operational cost of delay-sensitive workloads is less than the value they bring the operator. For this workload the operator is primarily interested in maximizing throughput. We further assume that the operator has provisioned the system to handle the peak of the delay-sensitive workload. The remainder of the paper explores how to schedule the delay-tolerant workloads to obtain maximum value to the operator while maintaining service guarantees for the delay-sensitive workload.

For our evaluation, we use a simple queueing model to approximate the response times and to calculate the resource usage.[2] Table 1 defines our notation. We assume that the system has a total (server) bandwidth of $B_{tot}$. The system bandwidth is divided into two partitions: a *primary partition* in which both delay-sensitive jobs and delay-tolerant jobs may be served, and a *secondary partition* in which only delay-tolerant jobs may be served. Our analysis of the primary partition assumes a generalized queuing model of a single server with a Poisson request rate of the delay-sensitive workload and the use of vacation periods [19] to serve the delay-tolerant workloads. Finally, we note that a series of infinitesimal vacation periods can be used to capture idle periods. During these vacation periods, neither delay-sensitive or delay-tolerant jobs are served, but the system stays ready to serve the next incoming delay-sensitive job.

## 3.2 Workload Model

To generalize our analysis, we assume each system can support $K$ delay-sensitive and $C$ delay-tolerant workload classes. In this section we describe the characteristics of each workload type.

**Delay-sensitive workloads:** In the general case, each delay-sensitive workload $k$ is assumed to have a time-varying request pattern, with request rate $\lambda_{k,t}$ at time $t$. For simplicity, we divide time into smaller intervals called *buckets*, for which the requests (approximately) follow a Poisson process within each bucket. Furthermore, workload $k$ is assumed to have a service demand $L_k$ (with a first and second moment $\overline{L_k}$ and $\overline{L_k^2}$, respectively). When multiple delay-sensitive workloads are used, we assume workloads with a lower index are higher priority.

**Delay-tolerant workloads:** We assume $C$ delay-tolerant workloads, each with a service demand of $S_c$ (with a first and second moment $\overline{S_c}$ and $\overline{S_c^2}$, respectively). These workloads are assumed to have much lower priority than any of the delay-sensitive workloads. We assume an infinite queue of delay-tolerant jobs; these are only served if no delay-sensitive jobs are queued and the server is not intentionally remaining idle.

**Idle periods:** To ensure that sufficient service guarantees can be provided, delay-tolerant jobs cannot always be scheduled whenever the server finishes a job and there are no outstanding delay-sensitive jobs pending. Instead, the server may have to remain idle for some time. For the purpose of our analysis, we assume that there is a third type of job, with infinitesimal job size ($S_* \to 0$). These jobs can be thought of as part of idle periods, during which there are no delay-sensitive jobs to serve and the server selects to not serve any delay-tolerant jobs. As the "idle jobs" are infinitesimal in size, they can be added without affecting the (average) waiting time $W_{k,t}$ of any delay-sensitive job of class $k$. We assume that an idle intensity $\phi_{*,t}$ can be selected such that on average the combined fraction of time in each bucket that the server is idle approaches $p_t^{idle}$, as $S_* \to 0$.

## 3.3 Important System Properties

**Service guarantees:** We assume that service guarantees are either expressed in terms of average response times or percentiles. With the former, we want to ensure that the average response time $R_{k,t}$ is always less than or equal to the response time target $R_k^m$ (i.e., $R_{k,t} \leq R_k^m, \forall t$). In practice, however, organizations often want to ensure that some large fraction (e.g., $\theta = 99.9\%$) of their customers get waiting times better than $W_k^\theta$. For this, we must ensure that the cumulative distribution function $F(W_{k,t})$ of the waiting times is greater than $\theta$ for $W_{k,t}^\theta$ (i.e., $F(W_{k,t}^\theta) \geq \theta$) [28].

**Utilization objective:** Based on our assumption that the selected delay-tolerant workloads provide value to the operator, the system should serve as many of these jobs as possible, providing the service guarantees of the delay-sensitive workloads are not affected. This corresponds to maximizing $\sum_t \left(\sum_{k=1}^K \lambda_{k,t}\overline{L_k} + \sum_{c=1}^C (\phi_{c,t}^f + \phi_{c,t}^g)\overline{S_c}\right)$, where $\phi_{c,t}^f$ and $\phi_{c,t}^g$ are the average job throughput of the delay-tolerant workload $c$ running on the primary and secondary partition, respectively.

**Non-preemptive:** Throughout this paper we do not consider preemption of the delay-tolerant jobs. Of course, if the delay-tolerant jobs can be (instantaneously) preempted, then running delay-tolerant workloads during (otherwise idle periods) should not affect the response times of delay-sensitive workloads.[3] Preemption of different classes of delay-sensitive workloads, and alternative optimization objectives are briefly discussed in Section 6.

## 3.4 High-level Policy Classes

Traditionally, many organizations run their delay-sensitive workloads on systems isolated from those running delay-tolerant workloads. In this work, we are interested in policies in which the two workload types (at least partially) share resources. To allow for analytic policy evaluation, we consider a single server system. However, while some of the analysis may be difficult to generalize, we believe that our results are more generally applicable. Therefore, we keep the policy classes as clean as possible so that they are easy to generalize.[4]

We consider two simple policies to establish baseline performance: *separated* and *shared*. The first policy divides the server into two partitions. This policy keeps the workloads completely separated, with the delay-sensitive workload assigned to the primary partition, and the delay-tolerant workload assigned to the secondary partition. With this policy, it is optimal for the system to allocate a bandwidth $B_{tot} - \sum_{c=1}^C \phi_c^g \overline{S_c}$ to the primary partition, and $\sum_{c=1}^C \phi_c^g \overline{S_c}$ to the secondary partition. This ensures that the size of the

[2]Following standard convention we refer to the *response time* as the combined *waiting time* and *service time*, where the waiting time is the time a request is queued in the system for service and the service time (or holding time) is the time the request is being served itself.

[3]For the case that preemption is possible, our models can be thought of as a way to capture the performance impact of the delay-sensitive jobs when the delay-tolerant load operates on the granularity with which preemption takes place.

[4]Nunez-Queija *et al.* [25] considered a similar set of policies in the context of a link in an ATM or IP network, that carries both streaming traffic and elastic traffic.

Table 1: Notation

| Parameter | Symbol | Definition |
|---|---|---|
| Server | $B_{tot}$ | Total server (i.e., system) bandwidth capacity |
| | $B_t$ | Server bandwidth for the primary partition at time $t$ |
| | $T$ | Number of time intervals (or buckets) |
| Delay-sensitive Workload | $K$ | Number of delay-sensitive workload classes, with priority order $1 \leq k \leq K$ |
| | $\lambda_{k,t}$ | Job generation rate of delay-sensitive workload $k$ at time $t$ |
| | $\overline{L_k}$ | Average service demand of delay-sensitive workload $k$ |
| | $\overline{L_k^2}$ | Second moment of service demand of delay-sensitive workload $k$ |
| | $W_{k,t}$ | Average waiting time of delay-sensitive workload $k$ at time $t$ |
| | $R_{k,t}$ | Average response time of the delay-sensitive workload $k$ at time $t$ |
| | $R_k^m$ | Target threshold of average response time of delay-sensitive workload $k$ |
| Delay-tolerant Workload | $C$ | Number of delay-tolerant workload classes, each with equal *low* priority |
| | $\phi_{c,t}^f$ | Average throughput of delay-tolerant workload $c$ running on primary partition at time $t$ |
| | $\phi_{c,t}^g$ | Average throughput of delay-tolerant workload $c$ running on secondary partition at time $t$ |
| | $f_c$ | Fraction of throughput of the delay-tolerant workload that should be of class $c$ |
| | $\overline{S_c}$ | Average service demand of delay-tolerant workload $c$ |
| | $\overline{S_c^2}$ | Second moment of service demand of delay-tolerant workload $c$ |
| Idle Workload | $\overline{S_*}$ | Average service demand of infinitesimal "idle" jobs |
| | $\overline{S_*^2}$ | Second moment of service demand of "idle" jobs |
| Percentile Analysis | $p_k$ | Probability of being in "sensitive" service state $k$ |
| | $q_c$ | Probability of being in "tolerant" service state $c$ |
| | $F(W_{k,t})$ | Cumulative distribution function (CDF) of the waiting times $W_{k,t}$ |
| | $W_k^\theta$ | Target threshold of the $\theta$-percentile waiting times of the sensitive workload $k$ |

primary partition is maximized, and the secondary partition, where there are no delay-sensitive jobs being served, is fully utilized.

The shared policy uses a single partition to which all server bandwidth is allocated. The delay-tolerant jobs are served only when there are no outstanding delay-sensitive jobs.

We also consider a *hybrid* policy. It uses two partitions, but allows a fraction $\frac{\phi_c^f}{\phi_c^f+\phi_c^g}$ of the delay-tolerant jobs to be served by the primary partition. The remaining fraction $\frac{\phi_c^g}{\phi_c^f+\phi_c^g}$ run in the secondary partition. In this case, we allocate $\sum_{c=1}^{C} \phi_c^g \overline{S_c}$ bandwidth to the secondary partition.

# 4. STEADY-STATE ANALYSIS

We begin our analysis by considering the steady-state case, where the request rate of the prioritized delay-sensitive workload is constant. For simplicity, we consider a single delay-sensitive workload (but allow for potentially many delay-tolerant background workloads). In the following we describe how we model the primary "shared" portion of the server bandwidth. For the secondary partition, we allocate sufficient bandwidth to satisfy the desired throughput; i.e., we require that the bandwidth for the primary partition is:

$$B = B_{tot} - \sum_{c=1}^{C} \phi_c^g \overline{S_c}. \quad (1)$$

We first consider the average response time for a general job size distribution of the delay-sensitive jobs. This analysis is followed by an investigation of the tail behavior of the waiting time distribution when the job-size distribution is assumed to be exponential.

## 4.1 Mean Value Guarantees

We first consider the average response time (including both the waiting and service times) for a single priority class (i.e., $K = 1$). Using known results for an M/G/1 queue with vacation periods, the expected response time can be expressed as [29]:

$$\overline{R} = \frac{\overline{L}}{B}\left(1 + \frac{\lambda\frac{\overline{L^2}}{B^2}}{2(1-\rho)}\right) + \frac{\overline{U^2}}{2\overline{U}}, \quad (2)$$

where $\rho = \frac{\lambda\overline{L}}{B}$ is the utilization due to the delay-sensitive workload, $\overline{U}$ is the average service duration and $\overline{U^2}$ is the second moment of the vacation periods. Again, in our model the vacation periods are used to serve delay-tolerant jobs, as well as to capture idle periods.

For the purpose of our analysis, we now derive a more explicit expression for $\frac{\overline{U^2}}{\overline{U}}$. We assume that after some time $T$, $N_c(T)$ jobs have been served of class $c$ and let "*" be used as the subscript of the arbitrarily small "idle" jobs. Then, we can rewrite $\frac{\overline{U^2}}{\overline{U}}$ as follows:

$$\begin{aligned}
\frac{\overline{U^2}}{\overline{U}} &= \lim_{T\to\infty} \frac{\frac{1}{\sum_c N_c(T)}\left(\sum_c \sum_{i=1}^{N_c(T)} (\frac{S_{c,i}^2}{B^2})\right)}{\frac{1}{\sum_c N_c(T)}\left(\sum_c \sum_{i=1}^{N_c(T)} \frac{S_{c,i}}{B}\right)} \\
&= \lim_{T\to\infty} \frac{\sum_{c=1}^{C} N_c(T)\frac{\overline{S_c^2}}{B^2} + N_*(T)\frac{\overline{S_*^2}}{B^2}}{\sum_{c=1}^{C} N_c(T)\frac{\overline{S_c}}{B} + N_*(T)\frac{\overline{S_*}}{B}} \\
&= \lim_{T\to\infty} \frac{T\sum_{c=1}^{C}\phi_c^f\frac{\overline{S_c^2}}{B^2}}{T(1-\frac{\lambda\overline{L}}{B})} = \frac{1}{(1-\rho)}\sum_{c=1}^{C}\phi_c^f\frac{\overline{S_c^2}}{B^2}. \quad (3)
\end{aligned}$$

In the first step, we cancel out the $\sum_c N_c(T)$ parts and

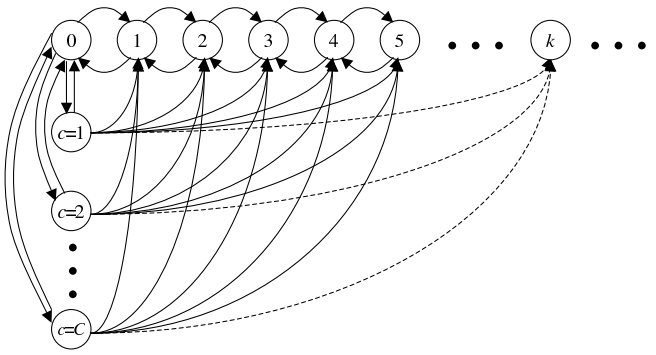



Figure 1: State transition diagram.

make use of the fact that the sums approach their expected values as $T \to \infty$. In the second step, we make use of the following properties: $\frac{N_*(T)\overline{S_*^2}}{\sum_{c=1}^{C} N_c(T)\overline{S_c^2}} = 0$[5], and $N_c(T) \to \phi_c^f T$ as $T \to \infty$. Also, any unused bandwidth for the delay-sensitive workload (i.e., $BT - \lambda TL$) is used for the delay-tolerant jobs (i.e., must be equal to $\sum_{c=1}^{C} N_c(T)\overline{S_c} + N_*(T)\overline{S_*}$, when $T \to \infty$). Finally, in the third step, we substitute $\rho = \lambda\overline{L}/B$. (The $T$ factors cancel out.) As a sanity check, we note that for the case of a single class $C = 1$ where the system is never idle (i.e., for which $\phi^f \frac{\overline{S}}{B} = 1 - \rho$), the above expression reduces to $\frac{\overline{S^2}/B^2}{\overline{S}/B}$.

## 4.2 Percentile Guarantees

We now turn our attention to service guarantees expressed with regards to how large a fraction $\theta$ of jobs are served faster than some threshold $W^\theta$. For this section we assume an exponential job-size distribution, and model the system as an M/M/1 system with vacation periods. While the exponential job-size distribution is needed to obtain a closed form waiting time distribution, we note that there may be room for similar approximate analysis approaches for more general job-size distributions.

### 4.2.1 State probabilities

Before deriving expressions for waiting time probabilities, we must first analyze the state probabilities for each system state. For this analysis we assume that service times are exponential. Figure 1 shows the state transition diagram for our system. Here, the states on the top row keep track of the number of delay-sensitive jobs currently in the queue (including the job in service) whenever there is a delay-sensitive job in service, with the $k = 0$ state corresponding to the idle state. The $c$ states corresponds to the cases in which a delay-tolerant job of class $c$ is in service. In the case of these jobs, we can accumulate more than one delay-sensitive job before changing state (at the end of a service period), as illustrated by the longer jumps to $k$ states. We use $p_k$ and $q_c$ to denote the probability the system is in states $k$ and $c$ respectively.

Consider first the flow going in and out of the $c$ states. The rate into these states (always from state $p_0$) is equal to $p_0\gamma f_c$, where $\gamma$ is the rate with which the server picks up a delay-tolerant job when idle and $f_c$ is the fraction of delay-tolerant jobs to be processed of class $c$. Similarly, the flow out of the state is equal to $q_c\frac{B}{\overline{S_c}}$. From these flow equations, it is easy to show that:

$$q_c = \frac{p_0\gamma f_c}{B/\overline{S_c}} = \frac{p_0\gamma f_c\overline{S_c}}{B} \quad (4)$$

Note that for the case when $\gamma \to \infty$, the probability $p_0$ of being idle approaches zero for any finite $q_c$, and when $\gamma \to 0$, the probability $q_c$ approaches zero (for any finite $p_0$).

Using vertical cuts in the above state transition diagram and considering the steady-state solution (i.e., for which the net flow across the cut is zero), we obtain the following equation for the state probabilities $p_k$:

$$\begin{aligned} p_k &= \frac{p_{k-1}\lambda\sum_{c=1}^{C} q_c\frac{B}{\overline{S_c}}\sum_{i=k}^{\infty} r_{c,i}}{B/\overline{L}} \\ &= p_{k-1}\rho + \sum_{c=1}^{C} q_c\frac{\overline{L}}{\overline{S_c}}\sum_{i=k}^{\infty} r_{c,i} \\ &= p_0\rho^k + p_0\sum_{c=1}^{C}\frac{\gamma f_c\overline{L}}{B}\sum_{j=1}^{k}\rho^{k-j}\left(1 - \sum_{i=0}^{j-1} r_{c,i}\right) \quad (5) \end{aligned}$$

where $\rho = \frac{\lambda\overline{L}}{B}$, and $r_{c,i} = e^{-\lambda\overline{S_c}/B}\frac{(\lambda\overline{S_c}/B)^i}{i!}$ approximates the probability that we have $i$ delay-sensitive jobs arriving during the service period of a delay-tolerant job of class $c$ (with average duration $\overline{S_c}/B$).

To calculate the above probabilities we need to solve for $p_0$. Before calculating $p_0$, we note that $\sum_{k=0}^{\infty}\rho^k = \frac{1}{1-\rho}$, and $\sum_{k=1}^{\infty}\sum_{j=1}^{k}\rho^{K-j}\sum_{i=j}^{\infty} r_{c,i} = \sum_{i=1}^{\infty} ir_{c,i}\sum_{j=0}^{\infty}\rho^j = \sum_{i=1}^{\infty}\frac{ir_{c,i}}{1-\rho} = \frac{E[r_c]}{1-\rho}$, where $E[r_c] = \sum_{i=1}^{\infty} ir_{c,i} = \lambda\frac{\overline{S_c}}{B}$ is the expected number of requests for delay-sensitive jobs during the service period of a delay-tolerant job, with average duration $\overline{S_c}/B$. With these observations in mind and the fact that $\sum_{c=1}^{C} q_c + \sum_{k=0}^{\infty} p_k = 1$, we can now solve for the idle probability:

$$p_0 = \frac{1}{\sum_{c=1}^{C}\frac{\gamma f_c\overline{S_c}}{B} + \frac{1}{1-\rho}(1 + \frac{\overline{L}}{B}\sum_{c=1}^{C}\gamma f_c E[r_c])}. \quad (6)$$

### 4.2.2 Waiting time distribution

We next derive expressions for the cumulative distribution function (CDF) of the waiting time distribution. To do this, we use the PASTA property (Poisson arrivals see time average) [30], and calculate the CDF as a weighted sum of the individual CDFs of requests arriving in the different states.

Let $F(W)$ be the CDF of the waiting times $W$, and $\rho = \frac{\lambda\overline{L}}{B}$ the average utilization due to the delay-tolerant workload. Then, we can approximate $F(W)$ as follows:

$$\begin{aligned} &\sum_{k=0}^{\infty} p_k\left(1 - \sum_{n=0}^{k-1} e^{-\frac{B}{\overline{L}}W}\frac{(\frac{B}{\overline{L}}W)^n}{n!}\right) \\ &+\sum_{c=1}^{C} q_c\sum_{k=1}^{\infty}\frac{r_{c,k}}{1 - r_{c,0}}\frac{1}{k}\sum_{i=1}^{k}\left(1 - \sum_{j=0}^{i} e^{-\frac{B}{\overline{L}}W}\frac{(\frac{B}{\overline{L}}W)^j}{j!}\right). \quad (7) \end{aligned}$$

Here, we have approximated the remaining service time of the $q_c$ states with the service time of a regular request. This allows us to approximate the combined remaining service time distribution (CDF) of the outstanding class $c$ job, plus

[5]To see this, we note that even though $p_{idle} = \frac{N_*(T)\overline{S_*}}{TB}$ must stay constant as we let $\overline{S_*} \to 0$ for the artificial "idle" vacation periods, in comparison to the finite $\overline{S_c^2}$, the term $\overline{S_*^2}$ decrease much faster in size, such that the condition holds true.

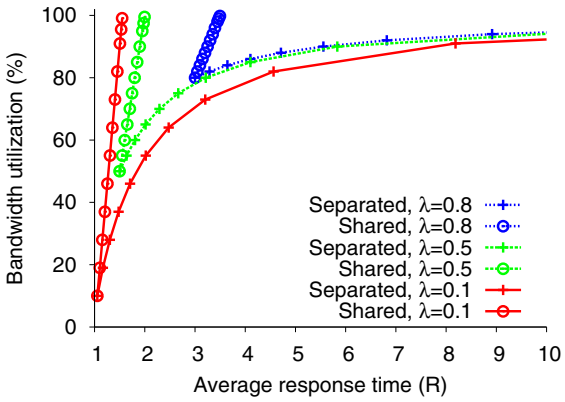



Figure 2: Example utilization and average service time under steady state. ($B_{tot} = 1, \overline{L} = \overline{L^2} = 1, \overline{S} = \overline{S^2} = 1$.)

the $i$ delay-sensitive jobs in front of the arriving job, using an Erlang distribution. This provides a reasonable approximation when the average service times are similar in size, and the assumption is conservative when delay-tolerant jobs are smaller. We show later in this section that small delay-sensitive jobs are advantageous to achieve good utilization, and such conservative operation points therefore may be desirable.

#### 4.2.3 *Percentile constraints*

Given a reasonable evaluation mechanism of the above expression, we can now solve the equation $F(W^\theta) = \theta$, where $\theta$ is the desired percentile and $W^\theta$ is the threshold for which we expect that the fraction $\theta$ of the requests will have no greater waiting time than.

A problem with the above expression is that it involves summations with an unbounded number of terms. To resolve this, we truncate the sums over $k$ at $k^*$. With the probabilities used within these sums, this results in an error no larger than:

$$1 - \sum_{k=0}^{k^*} p_k - \sum_{c=1}^{C} q_c \sum_{k=1}^{k^*} \frac{r_{c,k}}{1 - r_{c,0}}. \quad (8)$$

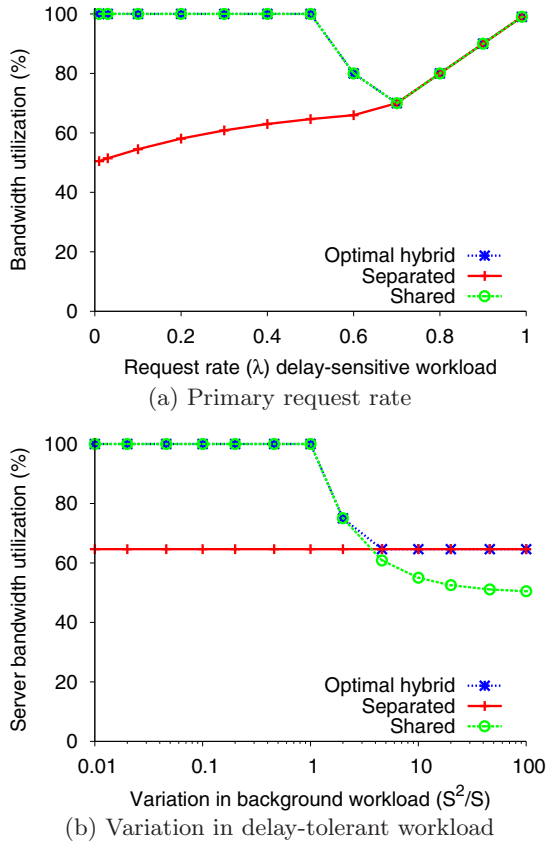
For our evaluation we pick $k^*$ large enough that the error is $\leq$0.01% of the reported value.

### 4.3 Policy Comparisons

We now apply the results from Sections 4.1 and 4.2 to compare the performance of the policies defined in Section 3.4. Without loss of generality, the unit of service rate is chosen to be equal to the total server bandwidth, and a unit of time is the time the server on average needs to process a single delay-sensitive job. This gives $B_{tot} = 1$ and $L = 1$.

#### 4.3.1 *Homogenous baseline scenario*

We first consider the utilization achieved using the two baseline policies: *separated* and *shared*. Figure 2 shows the utilization as a function of the average response times. Curves are shown for three different workload intensities: $\lambda$ = 0.1, 0.5, 0.8. With our normalized units this corresponds to the delay-sensitive load consuming 10%, 50%, and 80% of the total server resources, respectively. In these experiments we have a single delay-tolerant workload and assume




(a) Primary request rate

(b) Variation in delay-tolerant workload

Figure 3: Baseline comparisons using steady state analysis. ($B_{tot} = 1, R^m = 2, \lambda = 0.5, \overline{L} = \overline{L^2} = 1, \overline{S} = \overline{S^2} = 1$.)

that $\overline{L} = 1$, $\overline{L^2} = 1$, $\overline{S} = 1$, $\overline{S^2} = 1$ (our default values throughout the paper, unless otherwise specified.) With these parameters the server can achieve a utilization of close to 100%, without a significant increase in average response times using the *shared* policy ("o" points). In none of the three scenarios does the average response time increase by more than a factor of two. In contrast, with the *separated* policy ("+" points), the response times grow without bound.

Our initial experiment compares the ability of the baseline and optimal hybrid policies to improve the bandwidth utilizations of the server. In this experiment, a single delay-tolerant workload is used, and an average response time that does not exceed twice that of jobs served individually (i.e., $R^m = 2$) is maintained. In these experiments we vary the resources required by the delay-sensitive workload (by varying $\lambda$), and the variation in the job-size distribution of the delay-tolerant workload (by varying $\overline{S^2}$).

To obtain the optimal operation point of the shared policy, we rewrite the above problem formulation as a linear program (LP) in terms of $\phi_c^f$. Similarly, for the hybrid policy we rewrite the above problem formulation as a linear program (LP), for each possible partition $B$, in terms of $\phi_c^f$ and $\phi_c^g$. We can then use the solutions of these LPs to search for the best partition $B$.

When there is only a single delay-tolerant workload (in which the jobs can not be further categorized), the optimal

policy is to use the better of the two baseline policies (at each point of the parameter space). As we will show later, for more general cases this is not always the case.

Figure 3 shows the bandwidth utilizations of the baseline and *optimal hybrid* policies for a single delay-tolerant workload and an average response time that does not exceed twice that of jobs served individually (i.e., $R^m = 2$). In Figure 3(a) we vary the request rate of the delay-tolerant jobs, and in Figure 3(b) we vary the relative variation of the job-size distribution of the delay-tolerant jobs. These figures show that the *shared* policy can achieve very good utilization whenever the utilization due to the delay-sensitive workload itself is small (Figure 3(a)), or the variations in job sizes are sufficiently low (Figure 3(b)). The largest improvements (over *separated*) are achieved when the utilization due to the delay-sensitive workload itself is smaller. For example, with the same variation as for the delay-sensitive workload (i.e., with $\overline{S^2} = 1$) we obtain 100% utilization whenever $\lambda \leq 0.5$. While there is a drop in utilization for intermediate rates $\lambda$, we note that these utilizations can be improved upon if using delay-tolerant workloads with smaller variations in the job-size distribution.

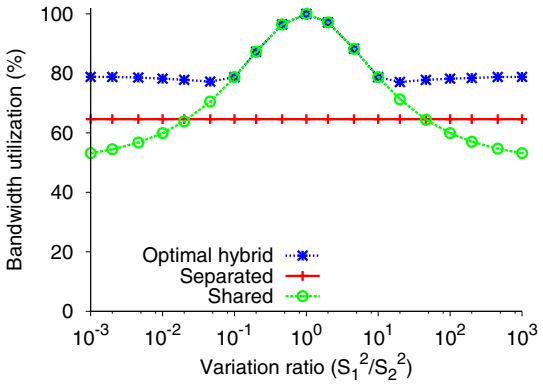
### 4.3.2 Job-size variations

As seen above, the variation in job-size distribution of the delay-tolerant jobs can have a significant impact on the achievable utilization levels. Figure 4 improves our understanding of the variation in job sizes. In particular, Figure 4 shows results using two different delay-tolerant workloads: both with the same average job size, but different variation. In Figures 4(a) and 4(b) we vary the ratio between the second moments of the two delay-tolerant workloads, and in Figure 4(c) we vary the percentage of jobs of the workload with small variation. Again, we note that the *shared* policy performs best when there are small variations in the job-size distribution; i.e., when either $\phi_1 << \phi_2$ in Figure 4(c) or $S_1^2 \approx S_2^2$ in Figures 4(a) and 4(b).

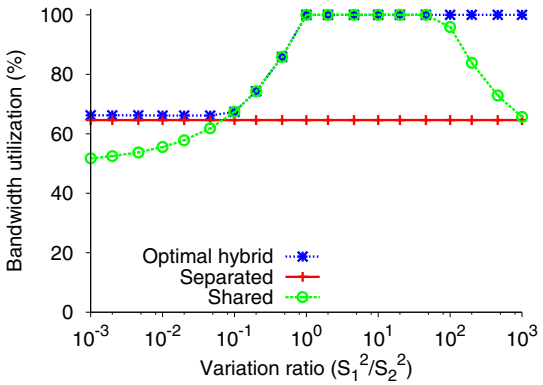
Perhaps the most telling example of when small job-size variations are beneficial is observed in Figure 4(b), where the high utilization on the right-hand-side of the graph is achieved when the dominating delay-tolerant workload class ($\phi_2$, with 90% of the background jobs) has much smaller variation than the other delay-tolerant workload class ($\phi_1$, with 10% of the background jobs). Similarly, in Figure 4(c) we can see that the utilization increases as the fraction of background load with small variance increases. This is intuitive, as smaller variations in the service times of these jobs ensures that there is a very small likelihood that a larger number of delay-sensitive jobs will get queued up behind such a job.

In constrast to the homogenous scenario (in Figure 3(b)), in all cases with two different delay-tolerant workloads (Figure 4), the hybrid policy outperforms both baseline policies for some range of the parameter space. In fact, there are regions in the parameter space in which we see an improvement of more than 30% in utilization (e.g., the right-most regions of the parameters spaces shown in Figures 4(b) and 4(c)). Much of this performance advantage comes from the hybrid policy's ability to separate the delay-tolerant jobs with high job-size variation from the rest of the workloads, while allowing the delay-tolerant jobs with low size variation to share resources with the delay-sensitive workload.

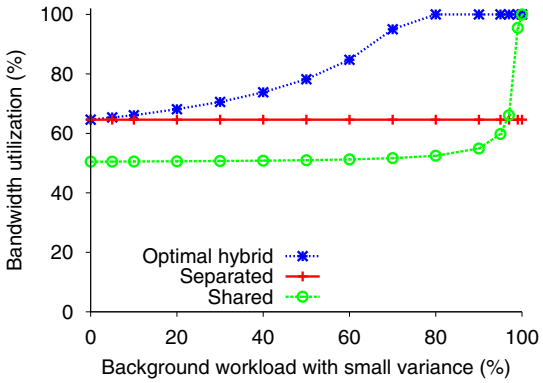
These examples illustrate the importance of understanding the workloads running on a system, and the value in separating the workloads with high variability from those with low variability. If jobs with low variability (in their job-size distribution) can be separated, then these jobs are great candidates to share resources with the delay-sensitive jobs. In contrast, any delay-tolerant workloads with high variability in job sizes should typically be processed separately. Overall, we believe that this illustrates the value in careful workload scheduling and server-resource management.




(a) $\frac{\phi_1}{\phi_1+\phi_2} = 50\%; \overline{S_1^2 S_2^2} = 1$




(b) $\frac{\phi_1}{\phi_1+\phi_2} = 10\%; \overline{S_1^2 S_2^2} = 1$




(c) $S_1^2 = 0.1, S_2^2 = 100$

Figure 4: Two class experiments. ($B_{tot} = 1, W^m = 2, \lambda = 0.5, \overline{L} = \overline{L^2} = 1, \overline{S_1} = \overline{S_2} = 1.$)

#### 4.3.3 Percentile service guarantees

An important consideration in our work is quantifying the effect that background workloads have on the tail of the waiting time distribution for the (prioritized) delay-sensitive workload.

Figure 5 shows the upper percentiles of the waiting time distributions, using both analytic approximations and simulation. Figure 5(a) shows the CDF of the waiting times in a system with only a delay-sensitive workload, and load intensities of $\lambda = 0.1, 0.5$, or 0.8. In contrast, Figure 5(b) shows results for a system running at 99% utilization, for various splits between delay-sensitive and delay-tolerant loads. We note that the analytic approximations typically are fairly accurate.

Figure 5(a) shows large variations in the waiting times as the intensity increases. This motivates operators (as has been noted before) to over-provision their systems to handle the peak loads. For example, to achieve a service guarantee (i.e., waiting time) of 8 for 99% of the jobs (or 3 for 90% of the jobs), the system's peak utilization must remain at or below 50% ($\lambda = 0.5$). Naturally, at lower loads (e.g., $\lambda = 0.1$) much stricter waiting time guarantees can be maintained, whereas at heavier loads the waiting times increase rather quickly.

Comparing the curves in Figures 5(a) and 5(b), it is interesting to note that the primary difference is that the y-intercepts have been pushed to zero. With the exception of scenarios with less delay-sensitive loads (e.g., the $\lambda = 0.1$ curves), adding the delay-tolerant loads does not have much effect on the waiting times of the delay-sensitive workloads. As the systems typically are sized with regards to satisfying the tail characteristics of the peak loads, our results indicate that as long as the lightly loaded curves stays on the left of the highly-loaded curves, the service guarantees are satisfied.

Based on the above observation, it is interesting to note that the tail typically is even less sensitive to running delay-tolerant jobs on the system than the average values. For systems using strict priority for delay-sensitive jobs, this in part is due to the service time of any request can at most be affected by one single delay-tolerant job (the job in service). In addition, if the service times of these delay-tolerant jobs typically are small (as is the case when $\overline{S}$ and $\overline{S^2}$ both are small, for example) not many other delay-sensitive jobs will have queued in the interim. The requests associated with the tail (i.e., requests that sees longer response times) are instead typically due to jobs arriving to a server with multiple queued delay-sensitive jobs. The probability of this occurring will not be greatly affected by the addition of delay-tolerant jobs being processed during periods when the system otherwise would be idle.

Figure 6 shows how well the baseline policies can improve system utilization. In these experiments we used a single delay-tolerant workload. We pick $S = L = 1$ to avoid approximation errors due to our Erlang assumption (Section 4.2.2) when jobs arrive to a system in which a delay-tolerant job is being served. With the *shared* policy the system can achieve a utilization of 100% with only a slight increase in the waiting times that can be guaranteed (10 for 99% and 5 for 90%, for $\lambda = 0.5$). For $\lambda = 0.1$ there are regions for which *separated* is better. For $\lambda = 0.8$, the waiting times in a system without any background jobs (as shown in Figure 5(a)) is higher for the 90th percentile than the full range used in Figure 6, and is therefore omitted.

# 5. TIME VARYING WORKLOADS

In this section, we extend the average delay analysis (based on an M/G/1 queuing system with vacation periods) from Section 4 to consider time-varying request rates for the delay-sensitive workload. This is typical of delay-sensitive workloads like Web servers [31]. We investigate the benefits of adaptive workload-management policies, with which the delay-tolerant workload can be shifted both between the partitions and with regards to the current request rate.

## 5.1 Time Generalization

As in the previous part of the paper we are interested in *maximizing* the overall utilization of the system. As before we assume a single server system with a total service rate $B_{tot}$, which can be partitioned into two parts: a primary partition with bandwidth $B(t)$ and a secondary partition with bandwidth $B_{tot} - B(t)$. Service times are equal to the service requirements divided by the bandwidth (or service rate) of the partition serving that job. For example, a primary job of size $L$ would have a service time $L/B$. While the overall utilization could be calculated exactly as a continuous integral over time, $\frac{1}{T}\int_0^T \frac{\lambda(t)\overline{L}+\sum_{c=1}^{C}(\phi_c^f(t)+\phi_c^g(t))\overline{S_c}}{B_{tot}}$, we use a discrete approximation

$$\frac{1}{T}\sum_{t=1}^{T}\frac{\lambda_t\overline{L}+\sum_{c=1}^{C}(\phi_{c,t}^f+\phi_{c,t}^g)\overline{S_c}}{B_{tot}}, \tag{9}$$

and assume that the expected response time in each bucket is small enough that it can be calculated independently. The above expression now becomes our objective function, and within each bucket we must ensure that the average response time $R_t$ is no greater than the threshold time $R^m$.

## 5.2 Optimization Generalization

Similar to Section 4, we can use an optimization formulation to find the best operation point of each policy class. In general we require that there must be sufficient bandwidth $B_t$ and $B_{tot} - B_t$ to serve the request in each of the two partitions; i.e., for feasibility we require that

$$\lambda_t\overline{L} + \sum_c \phi_{c,t}^f\overline{S_c} \leq B_t, \forall t \tag{10}$$

$$\sum_c \phi_{c,t}^g\overline{S_c} \leq B_{tot} - B_t, \forall t \tag{11}$$

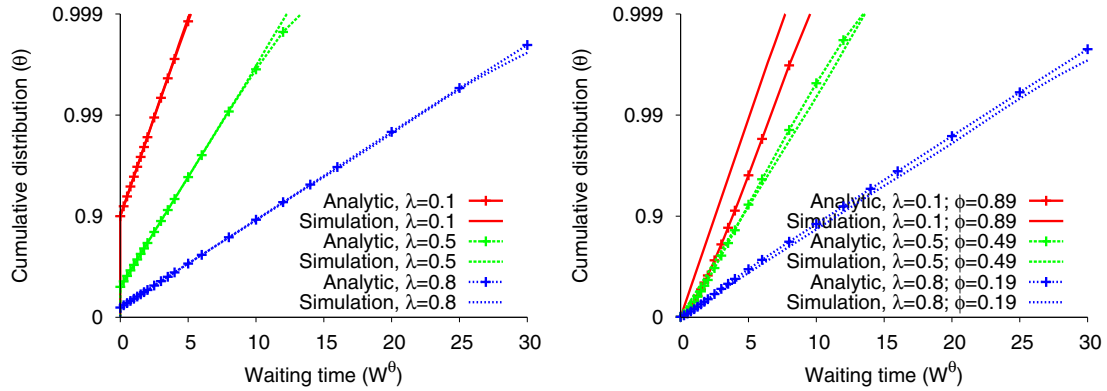
$$\phi_{c,t}^g \geq 0, \phi_{c,t}^f \geq 0, \forall c, t. \tag{12}$$

Furthermore, under the assumption that the primary partition operates as a M/G/1 server, we can generalize the expected response time expressions from Section 4 as follows:

$$R_t = \frac{\overline{L}}{B} + \frac{\lambda_t\frac{\overline{L^2}}{B_t^2}}{2(1-\rho_t)} + \frac{1}{2(1-\rho_t)}(\sum_c \phi_{c,t}^f\frac{\overline{S_c^2}}{B_t^2}), \tag{13}$$

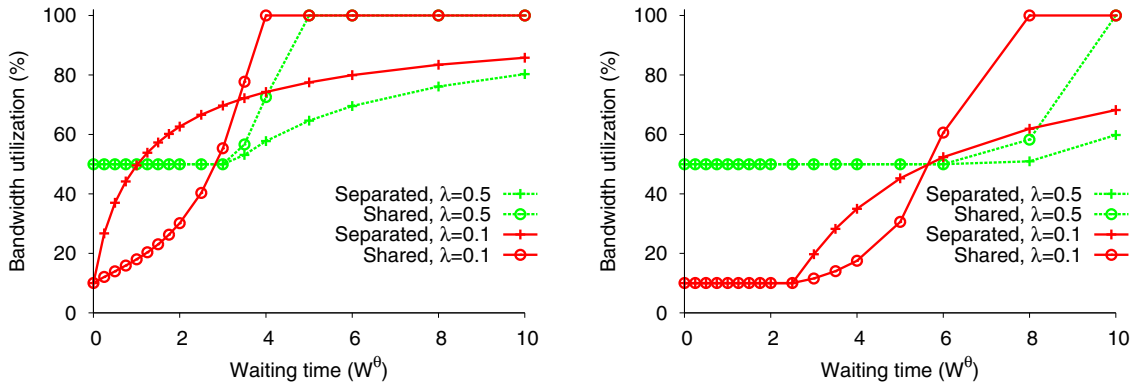
which must be less than $R^m$ (i.e., $R_t \leq R^m$).

Given a partition $B_t$ of the server bandwidth, the above (linear) constraints for the response times of the delay-sensitive workload ensure that our problem can be expressed as a linear program (LP) formulation with (9) as our objective function. Depending on the flexibility in terms of server




(a) Without background load ($\phi = 0$) (b) With background load (99% total utilization)

Figure 5: Cumulative distribution function (CDF) of the waiting times with and wihout any background (delay-tolerant) workload. ($B_{tot} = 1, \overline{L} = \overline{L^2}1, \overline{S} = \overline{S^2} = 1$.)




(a) $\theta = 90\%$ (b) $\theta = 99\%$

Figure 6: Utilization as a function of the 90-percentile and the 99-percentile service guarantees using the baseline policies. $B_{tot} = 1, \overline{L} = 1, \overline{S} = 1$.

bandwidth allocations between the two partitions, a different number of linear programs (LPs) must be solved to obtain the optimal policy within each policy class.

## 5.3 Baseline Policy Classes

A static policy is unlikely to be optimal across all points in time. In this section, we consider more advanced workload-scheduling policies to explore how to take full advantage of the characteristics of the delay-tolerant jobs and the time-varying characteristics of the delay-sensitive workload.

As established in Section 4, an opportunity exists to optimize the allocation of bandwidth to each partition, as well as the fraction of the delay-tolerant workloads to run in each partition. In this section we consider four base classes of policies. Interestingly, in the next section we show that performance close to the best of these policy classes can be achieved using a simple heuristic that utilizes the bandwidth partition of one of the simpler policies.

The policies considered differ in the freedom they allow along two dimensions: (i) adaptive vs. static bandwidth partitioning, and (ii) adaptive vs. static mix of the fraction of delay-tolerant jobs processed of each such class. In other words, does the policy allow $B_t$ and $\frac{\phi_{c_1,t}}{\phi_{c_2,t}}$ to vary with time, or not? Figure 7 illustrates our policy framework, and Figure 8 shows our four policy classes. Figures 8(c) and 8(d) show policies that use adaptive bandwidth split, and Figures 8(b) and 8(d) show policies that allow for adaptive delay-tolerant workloads (across time). In summary, we call the four policies: (a) Static bandwidth split with static background mix; (b) static bandwidth split with adaptive background mix; (c) Adaptive bandwidth split with static background mix; and (d) Adaptive bandwidth split with adaptive background mix.

## 5.4 Policy Comparisons

Motivated by the diurnal pattern of many existing workloads, we develop a synthetic time-varying workload to evaluate the policies described in Section 5.3. Specifically, we use a sinusoidal workload with a peak load of 60% and minimum load of 10%, resulting in a mean utilization of 35% (before any delay-tolerant workload is applied).[6] For this case, the response times at peak load are very close to two. In our analysis we therefore select $W^m = 2$ as our desired average service guarantee target.

[6]Also the utilization is selected to resemble that of some existing systems [1].

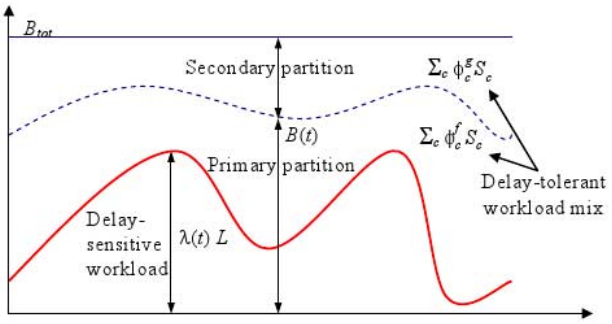



Figure 7: Policy framework for improving the effective utilization of a computer system.

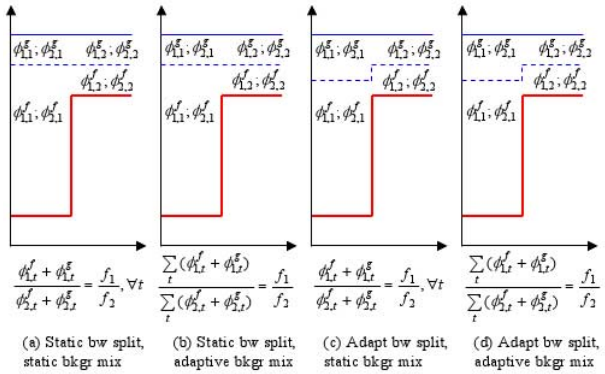



Figure 8: Example policies for simple workload scenarios.

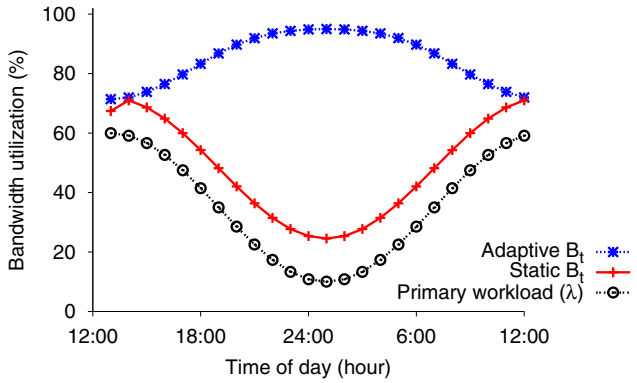



Figure 9: Scheduling around diurnal delay-sensitive workload. ($B_{tot} = 1, R^m = 2, \lambda = 0.35 + 0.25sin(\frac{t}{24}2\pi), \overline{L} = 1, \overline{L^2} = 1, \overline{S_1} = \overline{S_2} = 1, \overline{S_1^2} = 0.1, \overline{S_2^2} = 100.$)

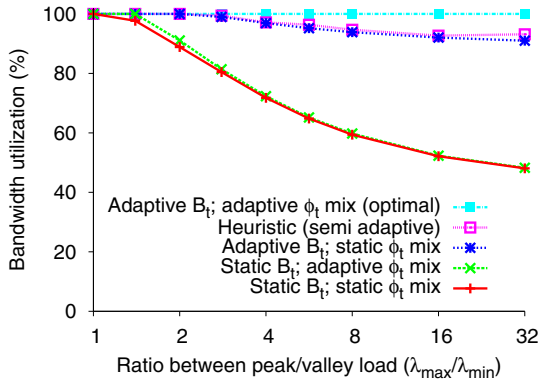
Figure 9 shows the results of the comparison of these different policies. The bottom curve shows the bandwidth consumed by the time-varying, delay-sensitive primary workload. The other two curves show the utilization with the best possible policy that adapts the bandwidth $B_t$ at each point in time, and the best possible policy that use a static partitioning for which $B_t$ is constant (equal to $B$), respectively. With both policies we optimize over $B_t$ (or $B$) and $\phi_{c,t}$, but keep the fraction of delay-tolerant jobs of each class $c$ constant over time (i.e., we limit the comparision to the static mix policies). In the particular example shown, we select the ratio to one, such that $\phi_{1,t}^f + \phi_{1,t}^g = \phi_{2,t}^f + \phi_{2,t}^g$ at each point in time.

Both policies achieve a significant improvement in utilization. The adaptive policy achieves an average utilization of 85% and the static policy an average utilization of 48%. Clearly, there is a significant advantage in using adaptive bandwidth partitioning. Such policies allow the system to better adapt to the time-varying characteristics of the workload demands, by adjusting both the bandwidth allocated to each partition, and the volume of delay-tolerant jobs allocated to each partition based on the current workload conditions.

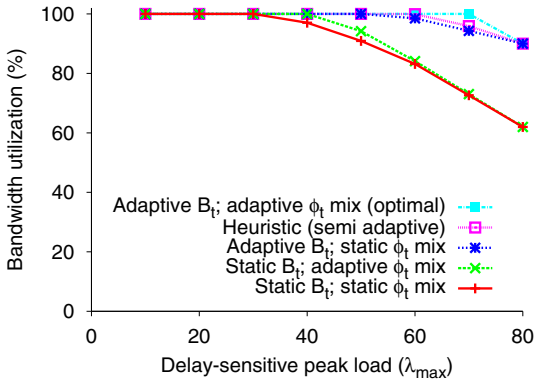
We now add the second dimension as well. In particular, we are interested to find out how much advantage may be gained from also adjusting the mix of the delay-tolerant jobs processed throughout the day. For the experiments in Figures 10(a) and 10(b) we vary the ratio between the maximum and the minimum load and the peak load itself, respectively. In the first example we set the average load to $\frac{\lambda_{max}+\lambda_{min}}{2} = 0.4$ and in the second example we pick a max-to-min ratio of 2. Since a peak of 80% is possible for this case, and the response times without any delay-tolerant workload for this extreme load is 3, we pick the response-time target for the interactive workload to $R^m = 3$ (rather than $R^m = 2$, which was used in the previous experiment).

While Figure 10 reveals some advantage in adjusting when to process small versus large (delay-tolerant) jobs over the course of a day, most of the advantage comes from adaptively changing the partitioning (and the fraction of jobs processed in each partition) over time. This can most easily be seen by comparing the achieved utilizations with the policies implementing some form of adaptive bandwidth allocations with the utilizations achieved with those restricted to static bandwidth allocation. This observation has important system implications, as it allows for simpler policies, which optimize based on current conditions, to reap most of the possible benefits.

There is, however, also some (small) advantage in higher-level optimization, such as scheduling jobs on a daily basis (e.g., small-variation delay-tolerant jobs during peak hours and large-variation jobs during less busy times). To take advantage of this, we consider a *heuristic policy* that greedily determines a bandwidth allocation one timeslot at a time, and then uses this bandwidth allocation when solving the above optimization formulation. For the higher peak loads (the right part of Figures 10(a) and 10(b)), this heuristic typically performs slightly better than the policy with a static mix, but not quite as well as the optimum. We believe the simplicity of the heuristic, as well as the *adaptive bandwidth split with static background mix* policy, together with a low performance penalty relative to the optimal policy, will make




(a) Varying $\frac{\lambda_{max}}{\lambda_{min}}$ ratio ($\frac{\lambda_{max}+\lambda_{min}}{2} = 0.4$)




(b) Varying peak load ($\frac{\lambda_{max}}{\lambda_{min}} = 2$)

Figure 10: Adaptive and static workload-management policies. ($B_{tot} = 1, R^m = 3, \lambda = (\lambda_{max}, \lambda_{min}), \overline{L} = 1, \overline{L^2} = 1, \overline{S_1} = \overline{S_2} = 1, \overline{S_1^2} = 0.1, \overline{S_2^2} = 100.$)

them desirable in systems with highly predictable diurnal workloads.

## 6. DISCUSSION AND EXTENSIONS

The optimization formulation in Section 5 and the framework used in this paper can be extended to more general cases in which there are multiple (prioritized) delay-sensitive workloads. Let $k$ be the index of the delay-sensitive workload with the k$^{th}$ highest priority.

**Non-preemptive, with K priority classes:** Based on Kella and Yechiali [29]:

$$R_{k,t} = \frac{L_k}{B_t} + \frac{\sum_{i=1}^{K} \lambda_{i,t}\overline{L_i^2} + (1-\rho_t)\frac{\overline{U_t^2}}{2U_t}}{2(1-\sigma_{k,t})(1-\sigma_{k-1,t})}, \qquad (14)$$

where $\sigma_{k,t} = \sum_{i=1}^{k} \lambda_{i,t}\frac{\overline{L_i}}{B}$ (with $\sigma_0 = 0$). Again, note that $\sigma_{k,t}$ and the denominator is independent of $\phi_{c,t}$ and the fraction $\frac{\overline{U_t^2}}{U_t}$ is the same linear function as above. This linearity allow for the use of similar LP formulations as used to obtain optimal mix and allocation of the delay-tolerant background workloads, between the partitions.

**Preemptive, with K priority classes:** Based on Kella and Yechiali [29]:

$$R_{k,t} = \frac{\overline{L_k}}{B_t} + \frac{\sum_{i=1}^{k} \lambda_{i,t}\overline{L_i^2} + (1-\rho)\frac{\overline{U_t^2}}{2U_t}}{2(1-\sigma_{k,t})(1-\sigma_{k-1,t})}. \qquad (15)$$

**Revenue Extension:** The above framework can also be used to consider alternative design objectives. For example, if each delay-sensitive workload has a revenue function $r_c$ per served job of class $c$ the objective function can be written as

$$\sum_t \sum_{c=1}^{C} (\phi_{c,t}^f + \phi_{c,t}^g) r_c \qquad (16)$$

where $r_c$ is the revenue associated with processing a job of class $c$. We note that as long as these alternative objective functions (and/or additional constraints) are linear with regards to the $\phi_{c,t}^f$ and $\phi_{c,t}^g$ terms, the same methodology can be used to obtain the optimal allocation of the delay-tolerant background workloads.

**Multi-server:** Larger servers can achieve higher utilizations by processing multiple jobs in parallel. For example, Karidis *et al.* [3] use an M/M/m queue (rather than multiple independent M/M/1 queues) to show the utilization improvements using multi-socket systems. We will explore more advanced models in future work.

## 7. CONCLUSIONS

In this paper we explored how to systematically improve the effective utilization of computer systems, while maintaining a desired responsiveness for delay-sensitive workloads. We demonstrate that by carefully managing resources, workloads, and scheduling delay-tolerant workloads around the higher priority workload(s), achieving our goal is feasible.

Our work leverages existing queuing theory of systems with vacation periods, and evaluates the responsiveness with regards to both average response times and on upper percentiles of the response time distribution, as values such as the 99th percentile often are important in practice. We show that by minimizing the variation in the size of tasks in the delay-tolerant workload, the mean response times in delay-sensitive workloads can remain adequately low even as the overall utilization of the computer system increases. Interestingly, the tail typically is even less sensitive to the addition of delay-tolerant jobs. This validates our intuition that the requests associated with the tail of the response time distribution are delayed by a queue of other delay-sensitive jobs rather than the servicing of delay-tolerant jobs. The delay-tolerant jobs are unlikely to affect the tail of the response time distribution for delay-sensitive jobs as service on a delay-tolerant job can only start if there are no queued delay-sensitive jobs.

We intend to extend our work in several ways. We plan to evaluate how well these theoretical properties hold in an actual system using real workloads. We will also investigate other pragmatic issues, such as how to extend our findings to multiple system resources.

# 8. REFERENCES


[1] L. Barroso, U. Hölzle, The case for energy-proportional computing, IEEE Computer 40 (12) (2007) 33–37.

[2] J. Hamilton, Where does the power go in high-scale data centers?, http://www.mvdirona.com/jrh/TalksAndPapers/JamesHamilton_SIGMETRICS2009.pdf (2009).

[3] J. Karidis, J. Moreira, J. Moreno, True value: Assessing and optimizing the cost of computing at the data center level, in: ACM Computer Frontiers, Ischia, Italy, 2009, pp. 185–192.

[4] B. Schroeder, M. Harchol-Balter, Web servers under overload: How scheduling can help, ACM Transactions on Internet Technology 6 (1) (2006) 20–52.

[5] G. Levine, The control of starvation, International Journal of General Systems 15 (2) (1989) 113–127.

[6] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, G. Jiang, Power and performance management of virtualized computing environments via lookahead control, Cluster Computing 12 (1) (2009) 1–15.

[7] X. Zhu, D. Young, B. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, L. Cherkasova, 1000 islands: an integrated approach to resource management for virtualized data centers, Cluster Computing 12 (1) (2009) 45–57.

[8] Y. Chen, D. Gmach, C. Hyser, Z. Wang, C. Bash, C. Hoover, S. Singhal, Integrated management of application performance, power and cooling in data centers, in: IEEE/IFIP NOMS, Osaka, Japan, 2010.

[9] A. Wierman, L. Andrew, A. Tang, Power-aware speed scaling in processor sharing systems, in: IEEE Infocom, Rio de Janeiro, Brazil, 2009, pp. 2007–2015.

[10] L. Andrew, M. Lin, A. Wierman, Optimality, fairness and robustness in speed scaling designs, in: ACM SIGMETRICS, New York, NY, 2010.

[11] A. Gandhi, M. Harchol-Balter, R. Das, C. Lefurgy, Optimal power allocation in server farms, in: ACM SIGMETRICS, Seattle, WA, 2009, pp. 157–168.

[12] D. Meisner, B. Gold, T. Wenisch, Powernap: Eliminating server idle power, in: Proc. ASPLOS, Washington, DC, 2009, pp. 205–216.

[13] K. Pruhs, P. Uthaisombut, G. Woeginger, Getting the best response for your erg, ACM Transactions on Algorithms 4 (3) (2008) 38:1–38:17.

[14] T. Abdelzaher, V. Sharma, C. Lu, A utilization bound for aperiodic tasks and priority driven scheduling, IEEE Transactions on Computers 53 (3) (2004) 334–350.

[15] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: Amazon's highly available key-value store, in: Proc. ACM SOSP, Stevenson, WA, 2007, pp. 205–220.

[16] D. Gaver, Probability models for multiprogramming computer systems, Journal of the ACM 14 (3) (1967) 423–438.

[17] E. Lazowska, The use of percentiles in modeling CPU service time distributions, in: Computer Performance, K. Chandy and M. Reiser (eds), North Holland Publishing Co., 1977, pp. 53–66.

[18] P. Bodik, C. Sutton, A. Fox, D. Patterson, M. Jordan, Response-time modeling for resource allocation and energy-informed slas, in: MLSys '07, Whistler, BC, 2007.

[19] B. Doshi, Queueing systems with vacations - a survey, Queueing Systems 1 (1986) 29–66.

[20] B. T. Doshi, Single server queues with vacations (1991).

[21] H. Takagi, Queueing analysis. a foundation of performance evaluation, volume 1: Vacation and priority systems, Elsevier Science Publishers (1991).

[22] E. Altman, K. Avrachenkov, U. Ayesta, A survey on discriminatory processor sharing, Queueing Systems 53 (2006) 53–63.

[23] S. Aalto, U. Ayesta, S. Borst, V. Misra, R. Nunez-Queija, Beyond processor sharing, SIGMETRICS Perform. Eval. Rev. 34 (4) (2007) 36–43.

[24] M. Harchol-Balter, T. Osogami, A. Scheller-Wolf, A. Wierman, Multi-server queueing systems with multiple priority classes, Queueing Systems 51 (2005) 331–360.

[25] R. Nunez-Queija, J. L. van den Berg, M. R. H. Mandjes, Performance evaluation of strategies for integration of elastic and stream traffic, Tech. rep., Amsterdam, Netherlands (1999).

[26] M. Litzkow, M. Livny, M. Mutka, Condor: a hunter of idle workstations, in: Proc. IEEE ICDCS, San Jose, CA, 1988, pp. 104–111.

[27] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tucke, Condor-g: a computational management agent for multi-institutional grids, Cluster Computing 5 (3) (2002) 237–246.

[28] R. Jain, The Art of Computer Systems Performance Analysis, Wiley, New York, NY, 1991.

[29] O. Kella, U. Yechiali, Priorities in M/G/1 queues with server vacations, Naval Research Logistics 35 (1) (1988) 23–34.

[30] R. Wolff, Poisson arrivals see time averages, Operations Research 30 (2) (1982) 223–231.

[31] M. Arlitt, C. Williamson, Internet web servers: Workload characterization and performance implications, IEEE/ACM Trans. on Networking 5 (5) (1997) 631–645.