Evaluation of Performance Tradeoffs in Scheduling Techniques for Mixed Workload Multimedia Servers

Leana Golubchik* John C.S. Lui† Edmundo de Souza e Silva‡ H. Richard Gail§

Abstract

Many modern applications can benefit from sharing of resources such as network bandwidth, disk bandwidth, and so on. In addition, it is desirable to store data that can be of use to many different classes of applications, e.g., digital libraries type systems. Part of the difficulty in efficient resource management of such systems can then occur when these applications have vastly different performance and quality-of-service (QoS) requirements as well as resource demand characteristics. In this work, we present a performance study of a multimedia storage system which serves multiple types of workloads, specifically a mixture of continuous and non-continuous workloads, by allowing sharing of resources among these different workloads while satisfying their performance requirements and QoS constraints. The broad aim of this work is to examine the issues and tradeoffs associated with mixing multiple workloads on the same server to explore the possibility of maintaining reasonable performance and QoS requirements without having to partition the resources. The main contribution of this work is the exposition of the tradeoffs involved in resource management in such systems. Although many different resources can be considered, here we concentrate mostly on the I/O bandwidth resource. The performance metrics of interest are the mean and variance of the response time for the non-continuous applications and the probability of missing a deadline for the continuous applications. The increased use of buffer space resources is also considered as a tradeoff for improvements in the above stated performance metrics.

1 Introduction

Many modern applications can benefit (cost-wise) from sharing resources such as network bandwidth and disk bandwidth. In addition, it is desirable to store data that can be of use to multiple classes of applications, e.g., digital libraries type systems. Efficient resource management is essential in providing scalability in large multimedia information systems serving a variety of applications,

*Corresponding author. Department of Computer Science, University of Maryland, College Park, MD 20742, U.S.A., (301) 405-2751 (phone), (301) 405-6707 (fax), leana@cs.umd.edu.

†Computer Science & Engineering Dept, CUHK. This research is supported in part by the Earmarked Grant CUHK4430/99E and the Mainline Research Grant.

‡CS Department, Federal University of Rio de Janeiro. This research is supported in part by grants from CNPq/ProTeM, CNPq, PRONEX and FAPERJ.

§IBM T.J. Watson Research Center.

where part of the difficulty is that these applications have vastly different performance and quality-of-service (QoS) requirements as well as resource demand characteristics.

One approach to dealing with this problem would be to simply share the resources among the different classes of requests with a best-effort attempt to meet the performance or quality-of-service (QoS) requirements of each. Another approach would be to partition the available resources between the different classes of workloads/requests, i.e., essentially maintain separate and independent servers. However, in general, "resource partitioning" is not a good idea, since one set of resources might remain idle while another set is overloaded. Furthermore, if copies of the same data are of use to multiple classes of applications, we may have to, in addition, incur a "penalty" for having to maintain consistency between multiple copies of the data. Thus, a more sensible approach would be to consider techniques which can share the resources among the different types of workloads while satisfying (to some degree) their performance requirements and QoS constraints.

In this paper, we consider a multimedia storage server which, in general, can service a variety of applications, requesting video, image, audio, and text data. For clarity and ease of exposition, we focus on two types of workloads: (1) continuous, and (2) non-continuous. For instance, the continuous requests (with continuity-type requirements) workload can correspond to requests for video streams whereas the non-continuous workload can correspond, for instance, to billing inquiries about the videos, thumbnail images corresponding to particular scenes in a video, and so on.

Clearly, the two types of workloads have different performance and QoS requirements. Specifically, the continuous workload requirements can be (a) low latency for starting a video display and (b) delivery of data at a particular rate (e.g., at 1.5 Mbps for an MPEG-1 stream) with little jitter, once the video display has been started[1]. On the other hand, the non-continuous workload, although it does not have jitter-type requirements, still requires reasonable response time — by *reasonable* we mean that it meets user-specified requirements, e.g., database applications, such a billing service, might require that $X\%$ of all transactions complete in under Y minutes.

In general, such a storage server can consist of some amount of processing capacity, storage capacity, and I/O transfer capacity, where the storage system is a multi-level hierarchy, e.g., including buffer space, disk-level storage, and tertiary storage. The cost/performance as well as scalability characteristics of such a system will be a function of the techniques used, at the various levels of the storage hierarchy, for data layout, scheduling of data retrieval, fault tolerance, caching[2] schemes, admission control, etc. However, to focus the discussion and to expose the tradeoffs involved in resource management in *mixed* workload systems, in this work we only consider the secondary storage and main memory levels of the storage hierarchy. More specifically, we do not focus on a particular architecture of the disk subsystem or specific data layout techniques but rather treat it as a "black box" with a certain maximum transfer capacity and overhead characteristics (i.e., seek, rotational latency, etc.), and instead concentrate on the scheduling of resources for use by both, continuous and non-continuous type requests.

[1]The low latency can also be viewed as high throughput, i.e., being able to sustain as many simultaneous video streams as possible, given a particular server architecture.

[2]Here we use "caching" as a general term, i.e., between any two levels of a storage hierarchy.

We begin with a survey of related work. There is a multitude of work on the design of continuous media servers, some of which (and by no means all) include [17, 1, 3, 18], where the authors mostly focus on data layout and retrieval and delivery techniques which facilitate maintaining of continuity in data delivery while providing either deterministic or statistical QoS guarantees. In [8, 9, 11, 10], authors present novel techniques for enhancing the performance and functionalities of VOD system. For example, techniques such as streams merging so as to increase the system serving capacity[8], the subband coding technique to maintain the load balancing feature for disk storage system[9], the threshold-based replication technique to resolve the scalability issue in a VOD system[11], as well as an approach for system sizing and resource allocation for a family of data sharing techniques in a VOD system[10]. Scheduling of mixed workloads has not received as much attention. A first detailed study (to our knowledge) of mixed workload scheduling issues was presented in [12], where the authors discuss a stochastic approach to QoS provisions to both types of workloads as well as present analytical models for computing the performance measures of interest. This is the work upon which we build here, in studying the tradeoffs involved in servicing mixed workloads. Recently, in an independent effort (from ours), in [13] the authors continue their work and propose more sophisticated (than in [12]) scheduling algorithms for servicing mixed workload. Hierarchical scheduling with a corresponding taxonomy of schemes is considered in [14].

In this paper, we present a performance study of a multimedia storage system which serves multiple types of workloads, and specifically a mixture of continuous and non-continuous workloads, by allowing sharing of resources among these different workloads while satisfying their performance requirements and QoS constraints. The main contribution of this work is the exposition of the *tradeoffs* involved in resource management in such systems. Although many different resources can be considered, here we concentrate mostly on the I/O bandwidth resource, where the performance metrics of interest are (a) the mean and variance of the response time for the non-continuous applications and (b) the probability of missing a deadline for the continuous applications. The main tradeoff here involves seek optimization opportunities which can either aid in or be detrimental to response time of non-continuous requests, depending on the workload experienced by the system. We argue that this tradeoff and its performance implications are best exposed through the technique of "mini-cycles", which is the main focus of this paper. Note that, this is a simple concept, which, e.g., has also been briefly mentioned in [13]; however, to the best of our knowledge no other work presents a systematic performance evaluation of the implications of this technique.

Moreover, we would like to point out that, to the best of our knowledge, all other works consider only low to moderate continuous workloads. In contrast, in our study, we investigate the performance tradeoffs under *high* (and specifically maximum possible) continuous workloads, in order to illustrate our points (since it often does not matter what resource management techniques are used at low loads). Furthermore, it is often desirable for cost-based reasons to run the storage server at a high (or maximum) number of continuous requests (provided that QoS requirements are satisfied), as long as reasonable response time to non-continuous requests can be provided. Thus, high continuous workloads do correspond to reasonable and important operating points at which

to consider our system[3]. Lastly, another contribution of our work is the consideration of use of buffer space resources as a tradeoff for improvements in the above stated performance metrics, i.e., response time (of non-continuous requests) and probability of missing deadlines (of continuous requests)[4].

The remainder of this paper is organized as follows. In Section 2 we review some background information. Section 3 describes the mixed workload scheduling algorithms, and Section 4 presents our performance model of the corresponding system. Section 5 discusses the results of our performance study of the scheduling algorithms. Our concluding remarks are given in Section 6.

2 General Approach and Background

In this section, we first review the basic concept of cycle-based (or group-based) scheduling [4, 17, 19], which is used for servicing *continuous* requests. We then discuss performance implications of *deterministic* vs. *statistical* QoS provisions for continuous requests, in order to motivate the use of a statistical technique in the remainder of the paper.

Cycle-Based Scheduling

In cycle-based scheduling algorithms, the retrieval of data from the disk sub-system, for servicing continuous requests, is performed on a cyclic basis where each cycle is of length T and in each cycle, the system retrieves data for N_c continuous requests. Note that in cycle-based scheduling algorithms, the transmission of data retrieved in the i^{th} cycle does not start until the beginning of the $(i+1)^{st}$ cycle[5]. This is motivated by the increased opportunities for performing seek optimization (i.e., data blocks needed for service are retrieved using a scan-type algorithm). The cost of this optimization is that the system needs additional buffer space to hold the retrieved data until the beginning of the next cycle. This cycle-based or (group-based) approach to servicing continuous streams is, for instance, suggested in [4, 17, 19], and the tradeoff between improved utilization of the disk bandwidth (due to seek optimization) and the need for additional buffer space is analyzed in several works, e.g., [4, 2, 19]. In general, larger values of N_c afford better seek optimization opportunities, but they also result in larger buffer space requirement.

Deterministic QoS provisions

One important design parameter in cycle-based scheduling is the actual value of the cycle length T. In general, the value of T is a function of the maximum number of continuous requests, N_c, that can be serviced by the system within a cycle and the degree of QoS that the system can provide.

[3] In any case, if it can be shown that reasonable response time can be provided to non-continuous requests at high (or maximum) continuous workloads, better response time to non-continuous requests can be obtained at lower continuous workloads.

[4] Use of buffer space resources as a tradeoff for seek optimization has been considered in [4] and [2], but in the context of continuous workloads only.

[5] That is, here we assume that the server is responsible for maintaining the continuity in data delivery, where the clients have relatively little buffer space. Thus, if the data delivery is not "offset" by one cycle from data retrieval, jitter may occur (due to seek optimization).

For instance, when insuring jitter-free retrieval/delivery of data, i.e., providing *deterministic (or worst-case) guarantees* [17, 2], T can be determined by considering the maximum time needed to retrieve N_c data blocks by scanning a disk. Formally, T can be expressed as:

$$T = \tau_{seek}^{max}(N_c) + N_c * \left(\tau_{rot}^{max} + \tau_{tfr}^{max}\right) \quad (1)$$

where $\tau_{seek}^{max}(N_c)$ refers to the worst case seek time for servicing N_c requests, i.e., when these N_c requests are uniformly distributed across the disk surface [6], τ_{rot}^{max} refers to the worst case rotational latency (i.e., a full rotation if we assume that the disk cannot support zero latency reads [15]), and τ_{tfr}^{max} refers to the worst case transfer time. Given a variable bit rate (VBR) stream (e.g, an MPEG-2 stream), the worst case transfer time can be determined by the peak display rate. Based on the above assumptions, the value of T will provide deterministic guarantee (e.g., all requests will receive service before the end of the cycle). However, the undesirable effect is that it can result in poor disk bandwidth utilization when there is a large deviation between the peak and the mean display rate.

Statistical QoS provisions

Another approach to determining the value of T is to provide *statistical guarantees*. That is, let τ_{N_c} be the random variable representing the service time of N_c continuous requests. Then, the system guarantees that the probability of the event where τ_{N_c} is greater than T is less than some predefined system parameter, e.g., p, that is:

$$\text{Prob}[\tau_{N_c} \geq T] \leq p \quad (2)$$

One can rely on the Chernoff's bound[6] theorem [7] (e.g., as in [13, 20]) to determine the value of T so as to service N_c requests under the probability constraint p. Let $F_{N_c}^*(s)$ be the Laplace transform for the random variable τ_{N_c} and let $F_{rot}^*(s)$ and $F_{tfr}^*(s)$ be the Laplace transforms for the random variables of rotational latency time and data transfer time, respectively. Since a cycle-based algorithm employs seek optimization and since the worst case seek time occurs when these N_c requests are equally spaced out on the disk surface [6], we have[7]:

$$F_{N_c}^*(s) = e^{(-s\,\tau_{seek}^{max}(N_c))}\left[F_{rot}^*(s)F_{tfr}^*(s)\right]^{N_c} \quad (3)$$

Let $M_{N_c}(s)$ be the moment generating function for the random variable τ_{N_c}. Since $M_{N_c}(s)$ is equal to $F_{N_c}^*(-s)$, applying Chernoff's theorem to bound the tail of the random variable τ_{N_c}, gives us the following:

$$\text{Prob}[\tau_{N_c} \geq T] = p \leq \inf_{\theta \geq 0}\left\{\frac{M_{N_c}(\theta)}{e^{\theta T}}\right\} \quad (4)$$

Note that the infimum (inf) operator applies to Equation (4), since, based on the Markov's inequality [7], that inequality holds for all $\theta \geq 0$. Therefore, the tightest bound is obtained by using

[6] Other approaches to bounding the tail of a sum of random variables exist; see [7] for details.

[7] For simplicity and clarity of exposition of the derivation, here we assumed a worst case seek of the disk scan as a function of N_c — hence the first term in Equation 3. Of course, the Chernoff bound applies to more general assumptions as well.

the *best* θ. Using standard numerical solution techniques, we can obtain the optimal θ^* which gives the tightest upper bound and thereby obtain the value of T. If τ_{N_c} is larger than T, then an *overflow* event occurs. In general, there are several approaches to handling the overflow situation. For example, the system can allow overflowing into the next cycle (i.e., finish servicing the requests in cycle i where, as a consequence, the N_c requests in cycle $i+1$ will have less than T time units to meet their deadlines). Or, the system can stop servicing the requests in the (overflowing) cycle i and proceed to service the next N_c continuous requests in cycle $i+1$. In this work, we only consider the first approach, i.e., that of allowing overflow into the next cycle. Specifically, in this work we will provide statistical guarantees of QoS for continuous requests, and thus an important performance measure under investigation here is the probability of overflow.

We will not focus on the merits of cycle-based (or group-based) scheduling any further here, as this is a well known approach to scheduling of continuous streams, but consider how to allocate whatever remains of the disk bandwidth resource to the non-continuous requests and focus on evaluating the *tradeoffs* between different approaches to doing this. In general, in our system, higher priority is given to continuous-type requests, i.e., we will guarantee, with a high probability of $(1-p)$ (as suggested above) that blocks for N_c continuous requests will be retrieved on-time (i.e., before time T), and without jitter. Due to statistical variations of service times among these N_c continuous requests, there will be time left in the cycle which is "not used" by the continuous requests. Therefore, the system can use this residual time to service non-continuous request. An interesting question here is, *how can we provide reasonable response time to non-continuous requests, given that (in a sense) higher priority is given to continuous requests.* We must first remind the reader that we investigate this issue under high system loads (i.e., maximum possible number of continuous requests that satisfies Equation (2)), which is an important distinction of our work. (We refer the reader to Section 1 for the details on the motivations for considering high system loads.) Therefore, in what follows, we consider a system which always has N_c continuous requests present.

3 Mixed Workload Scheduling

In this section we discuss the main tradeoffs in our system, in the context of three different scheduling algorithms, which explore the implications of: (1) ordering of service of non-continuous requests and (2) work conserving nature (or lack thereof) of the system.

Non-work conserving, first-come-first-serve (NW-FCFS)
The first simple algorithm is termed the non-work conserving, first-come-first-serve (NW-FCFS) algorithm. We have already described how the system schedules retrieval of continuous requests using cycle-based scheduling and scanning of the disk. Thus, whatever remains of the cycle will be used to service any non-continuous request present in the system, and these will be serviced in a FCFS manner. Formally, let there be two classes of requests in our system, class C (for continuous requests) and class NC for (non-continuous requests). Let N_c be the number of class C requests which can be fixed and then used to compute the cycle time T, as in Equation (4) above. Thus,

in the NW-FCFS algorithm, given a period of length T, we first serve N_c customers within the single period T. If after servicing all N_c customers the system still has some residual time within the period, that residual time is dedicated to servicing non-continuous requests in a FCFS manner. If there is not sufficient time remaining in T to service a non-continuous request, then the system idles until the end of that cycle. Note that this algorithm is non-work conserving in the sense that when some residual time exists and there are no non-continuous requests present, the system will not proceed to schedule the waiting continuous requests but rather, will remain idle until the end of the period[8]. An example of the NW-FCFS scheduling policy is depicted in Figure 1 where $N_c = 5$ and the overflow event is depicted in the second period.

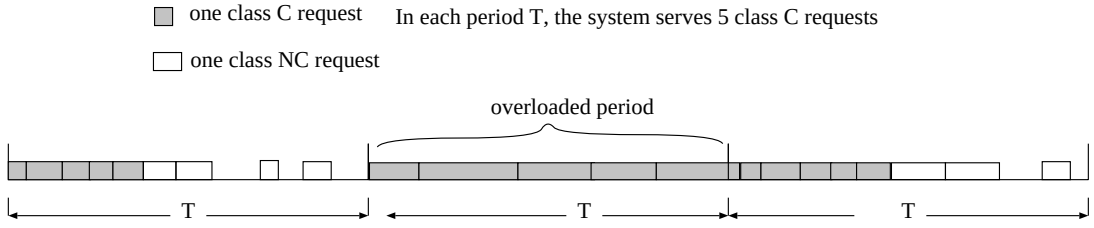


Figure 1: NW-FCFS Alg.: Within a cycle, serve class C first, then serve class NC in FCFS manner.

Although NW-FCFS is a possible algorithm for handling mixed workload requests, the resulting QoS (e.g., expected response time) for the non-continuous requests may be poor. For instance, if a non-continuous request arrives to the system at the beginning of a cycle, it has to wait until the system finishes the service of all continuous requests and only then, if there is some residual time in the cycle, the service of the non-continuous request can start. The situation will become worse if the arrival rate of the non-continuous requests is high which will result in long queueing delays and consequently further increase response time of the non-continuous requests.

To improve the response time of non-continuous requests, we can generalize the NW-FCFS algorithm, which we term the NW-FCFS(N_{mc}) algorithm as follows. Instead of having one cycle with length T, we divide this cycle into N_{mc} mini-cycles, each with a length of T/N_{mc}. Within each mini-cycle, the system uses the NW-FCFS algorithm to serve N_c/N_{mc} continuous requests[9]. That is, under this scheme, non-continuous requests can receive service if there is any residual time left at the end of each *mini-cycle*. It is important to note that we have two opposing effects here:

1. *By servicing all continuous requests within one scan, we achieve better disk bandwidth utilization because greater seek optimization opportunities exist for the continuous requests (which might result in more time available in a cycle for servicing non-continuous requests).*

2. *By servicing the continuous requests in many mini-cycles, we reduce the opportunities for seek optimization of the continuous requests, but we potentially improve the response time of the*

[8]The necessity to be non-work conserving is motivated by the need to maintain a specific rate of data delivery for continuous requests and that "early" data retrieval (i.e., earlier than is dictated by the desired delivery rate) will result in increasing growth in buffer space requirements.

[9]In practice, some earlier mini-cycle may serve $\lceil N_c/N_{mc} \rceil$ continuous requests while later mini-cycles may serve $\lfloor N_c/N_{mc} \rfloor$ continuous requests.

non-continuous requests (by servicing some of the earlier in a cycle). In addition, it is also possible that some continuous requests, especially those that are served in the last mini-cycle, will have a higher probability of missing deadline.

Therefore, these tradeoffs give room for an optimization problem. That is, how to find an *optimal* number of mini-cycles so as to minimize the response time of the non-continuous requests while providing the required QoS to the continuous requests. We will explore these tradeoffs and optimization later in the paper.

Another approach to further improve the response time of non-continuous requests is to service them in groups instead of in a FCFS manner, i.e., provide similar seek optimization opportunities (as in the continuous requests case) by servicing groups of non-continuous requests in a scan-type manner. We now describe a second algorithm, which we call "non-work conserving, gated (NW-Gated) algorithm", in terms of a single cycle of length T. Of course, the concept of mini-cycles can (and will) be applied to the NW-gated algorithm as well.

Non-work conserving, gated (NW-Gated)

The NW-Gated algorithm is similar to the NW-FCFS with the exception that instead of serving class NC requests in a FCFS manner, we serve them in a sorted order such that the total seek time is minimized (i.e., a form of SCAN). To achieve this, the system can organize all class NC requests and serve them according to their position with respect to the disk head, including the new class NC requests that arrive after the beginning of class NC service. However, this may introduce unacceptable delays for the requests that are far away from the disk head at the beginning of the service, since new requests that are closer to the head's position would have a higher priority of service. To avoid this problem, we use a *gated* discipline instead which is as follows. When the system completes service of the class C requests and there is still some residual time in a cycle, the system switches to servicing class NC requests currently present in the system. However, no new class NC requests are admitted into service until the current batch completes service (i.e., the gated discipline). If there is still time left in the current cycle, then the new class NC requests that arrived while the first batch was being served are eligible to start service. The gate is again closed and the process repeats until the end of the cycle. This discipline guarantees priority of old NC requests over new ones while trying to make efficient use of the disk, therefore, some notion of fairness is also provided. Similarly to the NW-FCFS, a non-continuous request is not serviced if there is not sufficient time to finish the service of that request before the end of the cycle. Lastly, this algorithm is classified as non-work conserving because it is possible that the server is idle while there are some class C requests waiting for service (i.e., when there are no non-continuous requests present in the system and there is still residual time in a cycle).

The NW-Gated algorithm is depicted in Figure 2. In this figure, the number of continuous requests that have to be served within a cycle, N_c, is equal to 5. Note that in the first cycle, the gate is closed three times, and each batch of non-continuous requests is served in a scan order so as to reduce seek overhead. In case there is an overloaded period (i.e., the second cycle), overflow is allowed into the next cycle.

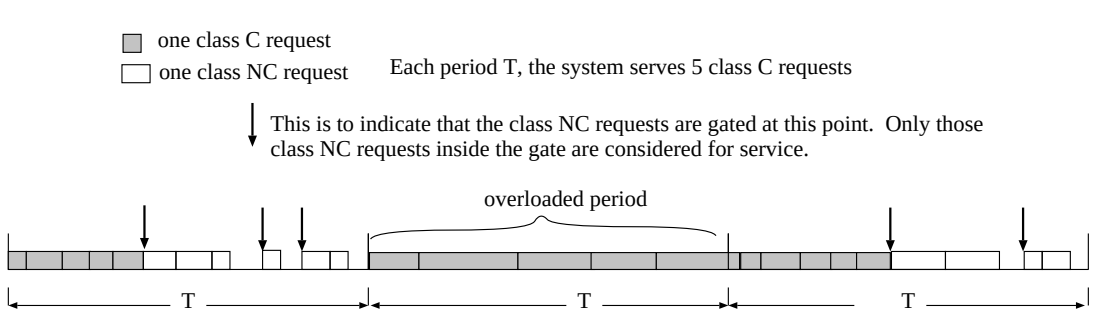


Figure 2: NW-Gated Alg.: Within a cycle, serve class C, then gated service for class NC.

As already mentioned, we can apply the concept of mini-cycles to this algorithm as well. For example, in the NW-Gated(N_{mc}) algorithm, there will be N_{mc} mini-cycles and the scheduling within a mini-cycle is similar to NW-Gated(1) algorithm as described above. Similar performance tradeoffs of disk bandwidth efficiency and response time of non-continuous requests exist in this class of algorithm and we will explore these tradeoffs later in the paper.

Of course the problem with the NW-FCFS(N_{mc}) and the NW-Gated(N_{mc}) algorithms is that idle times may be wasted at the end of each mini-cycle. To improve on these algorithms, we propose a third algorithm, which we term the pseudo-work conserving, gated (PW-Gated) algorithm[10].

Pseudo-work conserving, Gated (PW-Gated)

Assume we have N_{mc} mini-cycles. In the first mini-cycle, the system first starts servicing N_c/N_{mc} class C requests. At the end of this service, the system checks whether there are any class NC requests in the queue, if there is no class NC request waiting, then the system immediately begins the next mini-cycle. If there are some class NC requests, the system will serve these class NC requests in a gated fashion (as in the NW-Gated algorithm above). As before, if there is not sufficient time remaining in T to service a non-continuous request, then the system idles until the end of that cycle.

The server will switch back to servicing class C requests when either there are no more class NC requests in the queue or when the mini-cycle ends (whichever comes first). In either case, at that point a new mini-cycle begins. The system continues in this manner for the first $N_{mc} - 1$ mini-cycles. For the last mini-cycle, the system will behave like NW-Gated(1). This algorithm is pseudo-work conserving because during the first $N_{mc} - 1$ mini-cycles, the server is never idle and only in the last mini-cycle, there is a possibility of the server being idle (while there are continuous requests in the system). Another important point to observe here is that in this algorithm, the system aggressively services class C requests for the first $N_{mc} - 1$ cycles, therefore, it reduces the probability of overflow (i.e., missing the deadline) for class C requests that belong to the last mini-cycle (i.e., those serviced towards the end of the cycle) thereby reducing jitter in delivery. Figure 3 depicts the PW-Gated algorithm with $N_{mc} = 3$. In this figure, the system serves 5 class C requests in each mini-cycle. The system finishes the first mini-cycle early because there is no class NC

[10]Note that, when the number of mini-cycles is equal to 1, this algorithm behaves just like NW-Gated.

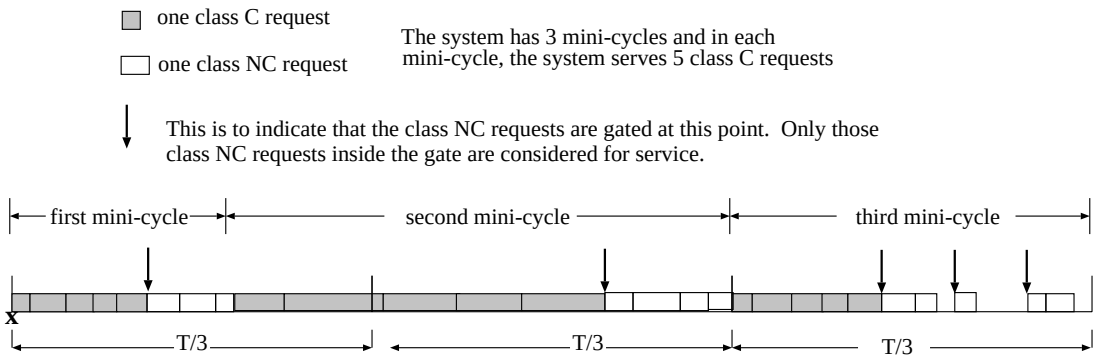


Figure 3: PW-Gated(3) Alg.: Serving class C requests, then gated service for class NC requests.

requests, and therefore, it switches to the second mini-cycle immediately. For the last mini-cycle, the system uses the NW-Gated algorithm to service class NC customers.

Remarks: Although there are many other possible algorithms for handling mixed workloads in storage systems, we do not present any more alternatives here, since that is not the focus of this work. More specifically, one could consider scheduling of requests whenever possible within a cycle (e.g., as in [13]), i.e., in a less *regular* manner. However, such approaches will not allow us to present a clear evaluation of the performance tradeoff, i.e., the performance consequences of various approaches to seek optimization, as stated in Section 1. Moreover, we would like to make the following observation. Thus far (as well as in Section 5), we have only considered performance characteristics of continuous requests on full cycle basis, i.e., the computation of probability of missing a deadline as well as the buffer space considerations (see Section 5) are discussed with respect to a full cycle of length T. However, observe that NW-FCFS and NW-Gated behave identically on all mini-cycles (in contrast to PW-Gated). Thus, we could consider assigning each continuous request to a specific mini-cycle within a cycle, i.e., always retrieve continuous request C_i during mini-cycle j, where $1 \leq i \leq N_c$ and $1 \leq j \leq N_{mc}$. Then, the transmission of data corresponding to C_i can begin in mini-cycle $j+1$, rather than being held in memory until the next cycle. This will reduce the buffer space requirements of continuous requests. However, the tradeoff here is that the probability of missing a deadline would have to be computed on a per-mini-cycle basis as well, rather than on per-cycle basis; this will surely increase in the probability of missing a deadline, although it is not clear how significantly. A *quantitative* assessment of this tradeoff is the topic of future work; however, note that such a tradeoff would not be possible, without the scheduling of requests in this more regular fashion, as suggested above.

4 Model

In this section we present a queueing model which will allow us to study the system described in Section 1 and the corresponding scheduling algorithms described in Section 3. We then present a discussion of the derivation of the workload parameters needed for this model.

4.1 Queueing Model

The model is depicted in Figure 4. It consists of two queues, Q_c and Q_{nc}, corresponding to

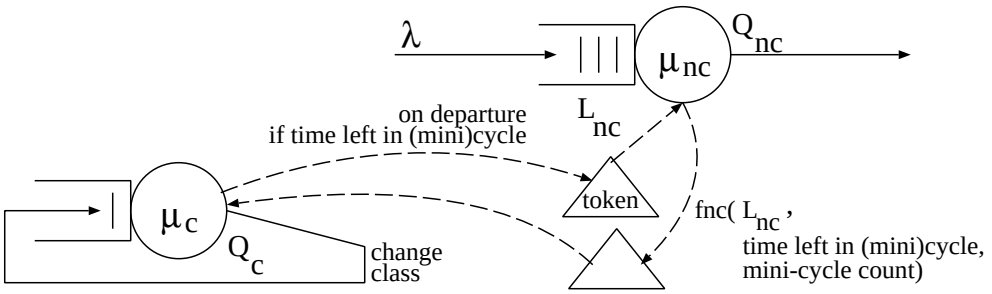


Figure 4: Queueing Model.

continuous and non-continuous requests, respectively. We now give an "informal" explanation of how this queueing system works; the more formal definition is given in the Appendix.

Let us, for ease of exposition, first assume that N_{mc}, the number of mini-cycles, is equal to 1. And, as already mentioned in the previous section, there is a notion of (global) cycle time, T, in the system and hence in the model as well. The queueing system behaves as follows. All N_c continuous requests are represented by a single customer of class C, which belongs to a closed chain [5], i.e., the customer is always present in the queueing system and after receiving service at Q_c, is routed back to Q_c. On the other hand, each of the N_{nc} non-continuous requests is represented by one customer of class NC. Customers of class NC arrive to queue Q_{nc} according to a Poisson process with a rate of λ, and depart from the system after receiving service at Q_{nc}, i.e., they belong to an open chain [5].

There is a single token in the system which, at the beginning of a cycle of length T, starts out at Q_c. Once the continuous customer completes service, on its departure, Q_c passes the token to Q_{nc} if there is still time left in the cycle. The continuous customer then comes back to Q_c but does not receive service until the token is returned to Q_c. If the service of the C customer is longer than the cycle time, T, by $t \geq 0$ time units, then the token remains at Q_c and the next cycles is of length $T - t$.

Operation of Q_{nc} is somewhat different. The server at Q_{nc} does not service any customers until it receives a token. Once the token is received, Q_{nc} begins servicing customers of class NC and continuous to do so until the cycle time expires, at which point it passes the token back to Q_c. Of course, if the cycle time has already expired when the token is received, it is passed back to Q_c immediately, without servicing any of the NC customers. The actual service discipline at Q_{nc} depends on the scheduling algorithm being modeled. Specifically, in the case of the NW-FCFS algorithm, the NC customers are serviced in a FCFS manner until the expiration of the cycle time. If the cycle time expires in the middle of servicing an NC customer, this customer's service is pre-empted and then initiated from the beginning (i.e., this is a non-resume type service) the next time the token comes back to Q_{nc}. In the case of the NW-Gated and the PW-Gated algorithms, the service disciple is more complex. Let L_{nc}^g be the number of NC customers at Q_{nc} when the

token arrives. These L^g_{nc} customers are serviced as a batch (i.e., using a SCAN-type algorithm as mentioned in Section 3). Q_{nc} continuous servicing the NC customers in this manner, i.e., in batches, where the size of the next batch is determined by the number of customers that arrive during the service of the current batch, until the cycle time expires. Again, as in the case of the NW-FCFS algorithm, if the cycle time expires in the middle of servicing an NC customer, this service is pre-empted and initiated from the beginning (i.e., not resumed) the next time the token returns to Q_{nc}.

Note that, in all three algorithms, even if the number of customers waiting at Q_{nc} goes to zero at some point, Q_{nc} will continue holding on to the token until the cycle time expires (this is the non-work-conserving part of these algorithms). Also note that, although this queueing model consists of multiple queues, the resources of interest (namely the I/O bandwidth) that this model represents are still shared (i.e., the sharing is done through the token).

To extend this model to multiple mini-cycles, i.e., $N_{mc} > 1$, the following modification must be made (the corresponding service times for each type of a customer are given in Section 4.2):

1. the single C customer now represents $\frac{N_c}{N_{mc}}$ continuous customers rather than all N_c; however we can still get away with only having a single C customer in the model, except that this single customer will change class [5] every time it departs from (and immediately returns to) Q_c — the (circular) numbering (or labeling) of classes will range from 1 to N_{mc} and we will now have the continuous customer alternating between N_{mc} classes, i.e., $C^1, C^2, \ldots, C^{N_{mc}}, C^1, \ldots$; this change of classes will represent going from one mini-cycle to the next[11]

2. the length of each mini-cycle is $\frac{T}{N_{mc}}$

3. the token-passing rules remain the same for NW-FCFS and NW-Gated except that the expiration of a mini-cycle (rather than a cycle) triggers passing of the token

4. the token-passing rules for PW-Gated in the last mini-cycle, i.e., the $(N_{mc})^{th}$ mini-cycle, remain the same

5. the token-passing rules for PW-Gated in mini-cycles $1, 2, \ldots, N_{mc} - 1$ are modified as follows. If $L^g_{nc} = 0$ either when the token arrives or after the service of a previous batch, then the token is passed back to Q_c immediately (this is the work-conserving part of the algorithm)

The queueing system described above is fairly complex. (Please see the Appendix for a more formal definition of the queueing model.) Thus, in Section 5 we present performance results obtained by it through simulation.

[11] Since the performance metric of interest for continuous requests is the probability of missing a deadline in a cycle, this model will suffice — continuous requests do not experience congestion in this system, i.e., in a sense they have priority over non-continuous requests.

4.2 Workload Characterization

Given the above model of the system, what remains is to discuss the service time distribution for both the continuous and the non-continuous customers[12]. Clearly, the service time of any customer in the model will correspond to some combination of seek, rotational latency, and transfer time (since we are modeling accesses to the disk sub-system). We begin our discussion with the seek time. In order to simplify the model somewhat, we will assume a deterministic seek, and derive an appropriate expression for each type of a request and each algorithm. (Note the distinction in the derivation below between seek time and seek distance; this distinction is due to the non-linearity of the seek time function as given in Table 1.)[13] Let CYL be the number of cylinders on the disk. Then,

1. the seek distance for the continuous customer (for all algorithms) will correspond to the worst case seek time, i.e., the seek time for servicing $\frac{N_c}{N_{mc}}$ requests uniformly distributed on the surface of the disk (see Section 2 for details); we will denote the seek distance corresponding to each continuous request by $d_{seek}^{max}(\frac{N_c}{N_{mc}})$, where

$$d_{seek}^{max}(\frac{N_c}{N_{mc}}) = \left\lceil \frac{CYL}{N_c/N_{mc}} \right\rceil \tag{5}$$

2. the seek distance for one non-continuous customer in the NW-FCFS algorithm will correspond to the average seek distance on a disk[6], i.e., $\left\lceil \frac{1}{3} d_{seek}^{full} \right\rceil$, where $d_{seek}^{full} = CYL$ denotes the maximum possible seek distance between any two cylinders on the disk (recall that in this algorithm the non-continuous customers are just serviced in a FCFS manner)

3. the seek distance for a gated group of non-continuous requests of size L_{nc}^g in the NW-Gated and the PW-Gated algorithms will be given by the following equation (recall that in these algorithms non-continuous customers are serviced in groups, and in a SCAN-type manner):

$$d_{seek}^{L_{nc}^g} = \left\lceil d_{seek}^{full} * \left[\left(\frac{2}{L_{nc}^g + 1} \right) \left(\frac{L_{nc}^g}{L_{nc}^g + 2} \right) + \left(\frac{L_{nc}^g - 1}{L_{nc}^g + 1} \right) \left(\frac{L_{nc}^g}{L_{nc}^g + 2} \right) \left(\frac{3}{2} \right) \right] \right\rceil \tag{6}$$

   where $d_{seek}^{full} = CYL$ denotes the longest possible seek distance (in number of cylinders) between any two cylinders on the disk

The intuition behind the derivation of Equation (6) is as follows. Firstly, note that, after finishing service of the continuous requests, the disk head is positioned in a random place on the disk, and thus, in general, the disk may have to service the L_{nc}^g requests by doing sweeps in both directions (one at at time, of course). That is, the L_{nc}^g requests are going to be serviced by sorting them based on the current location (i.e., where the disk head stopped after the last group of serviced requests) and direction (i.e., whichever direction the previous sweep was going in) of the disk head

[12] As already mentioned, the continuous customer belongs to a closed chain, and thus is always "present" in the system, whereas the arrival process of the non-continuous customers is Poisson with a rate of λ.

[13] The actual values of these seek times will depend on the disk parameters used — these will be given in Section 5 for the specific experiments discussed there.

and then retrieved in that order. Secondly, we make another simplifying assumption, namely that the L^g_{nc} customers together with the disk head, are uniformly distributed on the surface of the disk, i.e., they correspond to $L^g_{nc}+1$ equally spaced points that divide the disk surface into $L^g_{nc}+2$ partitions of equal size. Then, the first term in Equation (6) represents all the cases where the disk head corresponds to a point on an edge of the disk surface — in such cases, the disk head needs to seek a distance of $\frac{L^g_{nc}}{L^g_{nc}+2} d^{full}_{seek}$ to service these L^g_{nc} requests, and these cases occur with probability $\left(\frac{2}{L^g_{nc}+1}\right)$. The second term in Equation (6) represents all the cases where the disk is not on an edge of the disk surface — in such cases, the disk head needs to seek a distance of $(\frac{3}{2})(\frac{L^g_{nc}}{L^g_{nc}+2}) d^{full}_{seek}$, and these cases occur with probability $\left(\frac{L^g_{nc}-1}{L^g_{nc}+1}\right)$.

Given the seek distances derived above, the service time of the continuous customer, for all three algorithms, is as follows:

$$S_c = \tau^{edge}_{seek} + \frac{N_c}{N_{mc}} * \left[seek\left(d^{max}_{seek}(\frac{N_c}{N_{mc}}) \right) + \tau^c_{rot} + \tau^c_{tfr} \right] \tag{7}$$

where τ^{edge}_{seek} is an extra seek needed to get to an edge of a disk surface in order to do the SCAN corresponding to service of continuous requests[14]. The $seek(d)$ is the seek function which gives the seek time as a function of the seek distance (in number of cylinders). The specific function that should be used depends on the type of disks used in the system; the seek function for the type of disk used in our experiments (a Seagate Barracuda) is given in Table 1. The τ^c_{rot} and τ^c_{tfr} are random variables corresponding to rotational latency and transfer time portions of the service time of each continuous request (their distributions, for the experiments discussed in this paper, are given in Section 5), and $d^{max}_{seek}(\frac{N_c}{N_{mc}})$ is given in Equation (5). Recall that the single continuous customer in our model represents all the continuous requests being serviced in a single mini-cycle in the system. Hence, the $\frac{N_c}{N_{mc}}$ term in Equation (7).

The service time of each non-continuous customer in the NW-FCFS algorithm is:

$$S^1_{nc} = seek\left(\left\lceil \frac{1}{3} d^{full}_{seek} \right\rceil \right) + \tau^{nc}_{rot} + \tau^{nc}_{tfr} \tag{8}$$

where τ^{nc}_{rot} and τ^{nc}_{tfr} are random variables corresponding to rotational latency and transfer time portions of the service time of each non-continuous request (their distributions, for the experiments discussed in this paper, are given in Section 5), and $d^{full}_{seek} = CYL$. The $seek(d)$ function is the same as the one above.

The service time of one non-continuous customer, in a gated group of size L^g_{nc}, in the NW-Gated and PW-Gated algorithms is:

$$S^2_{nc} = seek\left(\left\lceil \frac{d^{L^g_{nc}}_{seek}}{L^g_{nc}} \right\rceil \right) + \tau^{nc}_{rot} + \tau^{nc}_{tfr} \tag{9}$$

where τ^{nc}_{rot} and τ^{nc}_{tfr} are as above, i.e., random variables corresponding to rotational latency and transfer time portions of the service time of each non-continuous request whose distributions, for

[14]Note that, this "extra" seek should not be "counted" in the response time of non-continuous requests; thus we include it here.

the experiments discussed in this paper, are given in Section 5. The $seek(d)$ is the same seek function as above, and $d_{seek}^{L_{nc}^g}$ is given in Equation (6).

5 Results

In this section, we examine the tradeoffs associated with service of mixed workloads by evaluating the performance of the scheduling approaches which were described in Section 3 via a simulation study. We first define the performance metrics used in this study. Let T_{nc}^i be the random variable representing the response time of the i^{th} non-continuous request. The performance metrics used for the non-continuous requests are $E[T_{nc}]$ and σ_{nc}, the expected response time and the variance of the response time, respectively:

$$E[T_{nc}] = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} E[T_{nc}^i] \qquad \text{and} \qquad \sigma_{nc} = \lim_{n\to\infty} \frac{1}{n-1} \sum_{i=1}^{n} \Big(T_{nc}^i - E[T_{nc}]\Big)^2 \tag{10}$$

Let E_{md}^i be the event that the i^{th} continuous request has missed the deadline[15] and E_{ol}^i be the event that the i^{th} cycle is overloaded. The performance metrics used for the continuous requests are P_{md} and P_{ol}, the probability of a continuous request missing a deadline and the probability of an overloaded cycle, respectively:

$$P_{md} = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} 1\{E_{md}^i\} \qquad \text{and} \qquad P_{ol} = \lim_{k\to\infty} \frac{1}{k} \sum_{j=1}^{k} 1\{E_{ol}^j\} \tag{11}$$

where $1\{f\} = 1$ if f is true and 0 if f is false.

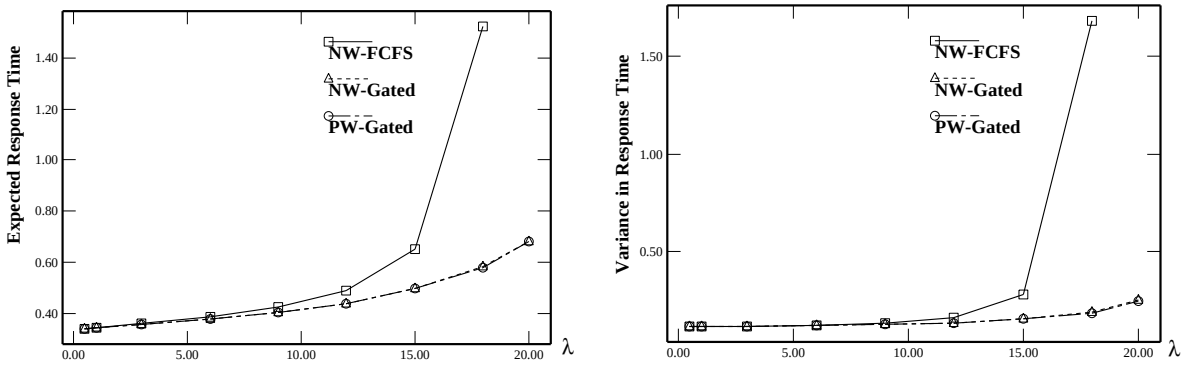
Table 1 gives the relevant characteristics of a typical Seagate disk used in the experiments presented here. Note that the seek time is in seconds, and it is a function of the request seek

Disk Capacity	2.25 GBytes
Number of cylinders (CYL)	5288
Transfer rate	75 Mbps
Maximum rotational latency	8.33 milliseconds
seek time function (secs)	$seek(d) = \begin{cases} 0.6 * 10^{-3} + 0.3 * 10^{-3} * \sqrt{d} & \text{if } d < 400 \\ 5.75 * 10^{-3} + 0.002 * 10^{-3} * d & \text{if } d \geq 400 \end{cases}$

Table 1: Seagate Barracuda 4LP Disk Parameters

distance d. Unless otherwise stated, we use the following parameters in the experiments presented in this section. We set a requirement that a disk in our system has to support $N_c = 24$ continuous requests, which represent MPEG streams with an average display rate of 1.5 Mbps each. The reason that we set $N_c = 24$ is to make sure that the system has to support a *high* continuous workloads. For the non-continuous request, the arrival process is modeled as a Poisson process

[15] Recall that, a missed deadline occurs when the server finishes service of a continuous request after a cycle of length T has already ended.



$E[T_{nc}]$ for $T = 1.48754$, $N_{mc} = 1$.

σ_{nc} for $T = 1.48754$, $N_{mc} = 1$.

Figure 5: $E[T_{nc}]$ and σ_{nc} for NW-FCFS, NW-Gated and PW-Gated under $T = 1.48754, N_{mc} = 1$.

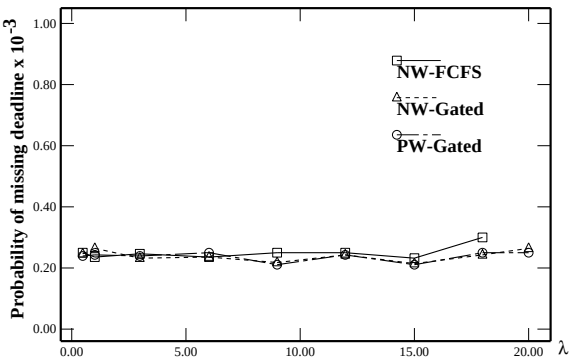
with rate λ, and the transfer size is modeled by an exponential distribution with mean 46.875 KBytes[16]. We assume that in all experiments (unless otherwise stated), the requirement is that the probability of an overloaded cycle has to be less than or equal to 0.01 (i.e., Prob$[\tau_{N_c} \geq T] \leq 0.01$). Therefore, we can guarantee that at most 1% of the service periods are overloaded. Given the disk parameters and the workload requirements, we can solve Equation (4) numerically to determine the value of T such that Prob$[\tau_{N_c} \geq T] \leq 0.01$. Based on this calculation, we have $T = 1.48754$ seconds and we set the value of T in all experiments (unless otherwise stated) to be this value so as to satisfy the QoS constraint. We model the transfer size of a continuous request as an exponential distribution with mean 2.25 Mbits (i.e., so that on the average, the display rate per stream per cycle is approximately 1.5 Mbps as for an MPEG1 stream). The rotational latency for both the continuous and the non-continuous requests is modeled as a uniform distribution in the range of $[0, 8.33]$ milliseconds[17].

5.1 Experiment 1: Comparison of scheduling algorithms under $N_{mc} = 1$

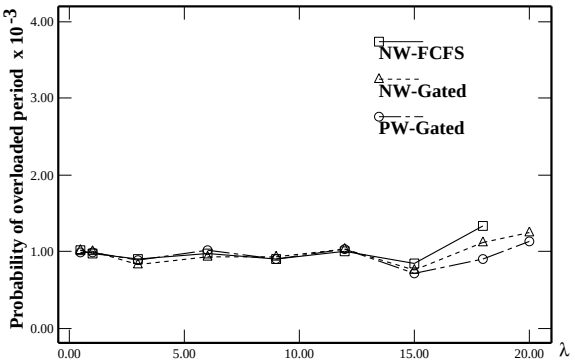
This experiment illustrates the benefits of seek optimization through simple re-ordering of non-continuous requests, i.e., the benefits of "Gated" service. To isolate this issue, we set the number of mini-cycles, N_{mc}, to 1 ($T = 1.48754$ is set at seconds). Figure 5 illustrates $E[T_{nc}]$ and σ_{nc} of non-continuous requests for the three scheduling algorithms, where both, the NW-Gated and the PW-Gated algorithms have a much lower $E[T_{nc}]$ and σ_{nc} for the non-continuous requests under moderate to high arrival rates. The greater efficiency in resource utilization of the Gated algorithms is clearly illustrated in Figure 5, where a system using a Gated algorithm is able to sustain higher workloads (than a FCFS system) while remaining stable, i.e., a FCFS system "blows up", or starts

[16] These simple distributions are used here for clarity of presentation of ideas; they suffice for our purpose, i.e., that of illustrating *tradeoffs* and performance *trends* rather than predicting *exact* performance numbers for the overall system.

[17] It is important to note that although for more advanced disks, the disk parameters are different than the one we use in Table 1. However, the performance and conclusion of this study remains valid.



P_{md} for $T = 1.48754$, $N_{mc} = 1$.



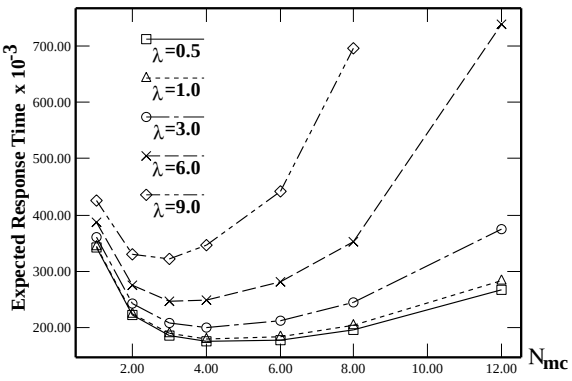
P_{ol} for $T = 1.48754$, $N_{mc} = 1$.

Figure 6: P_{md} and P_{ol} for NW-FCFS, NW-Gated and PW-Gated under $T = 1.48754$, $N_{mc} = 1$.

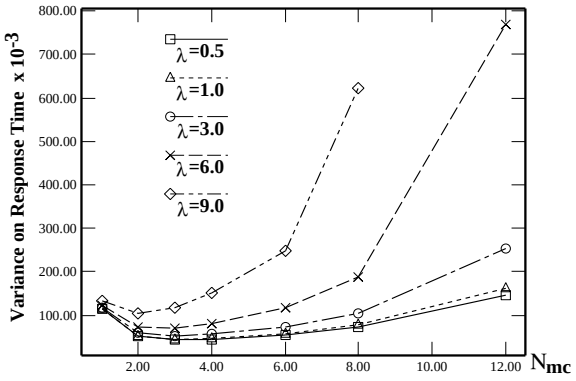
experiencing extremely long response times, at lower workloads than the Gated systems. The reason that the NW-FCFS has the worst performance at high non-continuous request workload is that many non-continuous requests are waiting in the queue and that this workload has a higher service time requirement than the gated algorithms. The gated algorithms are able to "reduce" the non-continuous requests' workload by re-ordering their service ordering and thereby reducing the aggregate seek times for the non-continuous requests. Finally, Figure 6 illustrates P_{md}, the probability of missing deadline, and P_{ol}, the probability of an overloaded cycle, for all scheduling algorithms. Observe that all three algorithms studied here perform within the quality of service (QoS) requirements set above, i.e., Prob$[\tau_{N_c} \geq T] \leq 0.01$.

5.2 Experiment 2: Convexity property of mini-cycle under NW-FCFS scheduling algorithm

In this experiment, we study the effect of mini-cycles on the performance of continuous and non-continuous requests under the NW-FCFS algorithm. Figure 7 illustrates the corresponding changes in T_{nc} and σ_{nc}, where we observe a *convexity property* — that is, when we initially increase the number of mini-cycles, there is an improvement in $E[T_{nc}]$ and σ_{nc}, which is due to the fact that with more (and therefore shorter) mini-cycles the waiting time (before the server switches back to servicing non-continuous requests) is reduced. However, there is a corresponding loss in seek optimization opportunities (due to fewer number of continuous requests being serviced in a mini-cycle). As the number of mini-cycles is increased further, we experience diminishing returns in performance gains due to reduced waiting time. At which time, the loss of bandwidth efficiency (due to loss of opportunities for seek optimization) becomes the dominant factor, and thus finally results in the reduced performance of the non-continuous requests. This point is further supported by the fact that, in Figure 7, as the load increases (i.e., higher values of λ) better performance is achieved with fewer mini-cycles (or conversely longer mini-cycles) — that is, as the load increases, the increased efficiency due to greater opportunities for seek optimization becomes a more important



$E[T_{nc}]$ for NW-FCFS with varying N_{mc}.



σ_{nc} for NW-FCFS with varying N_{mc}.

Figure 7: $E[T_{nc}]$ and σ_{nc} for NW-FCFS with varying number of mini-cycles and $T = 1.48754$.

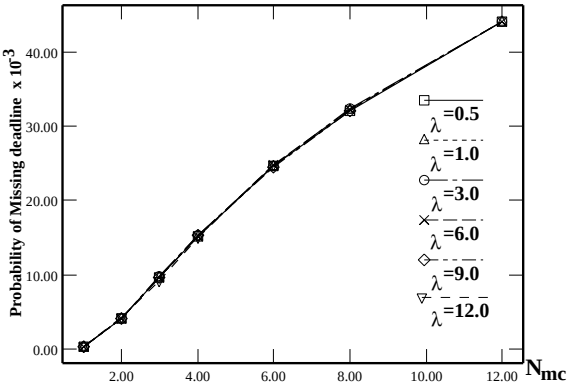


Figure 8: P_{md} for continuous requests under NW-FCFS under different values of mini-cycles.

factor. Another important point to observe here is that this *convexity property* implies that there exists an optimal operating point, i.e., an N^*_{mc}, which corresponds to the *optimal* number of mini-cycles[18] that will minimize $E[T_{nc}]$ and σ_{nc}, and that is where we should be operating our system.

Although we can reduce $E[T_{nc}]$ and σ_{nc} for the non-continuous requests by operating the system at $N_{mc} = 4$, the NW-FCFS algorithm may not be able to support, at this operating point, the QoS required by the continuous requests. Figure 8, illustrates that the probability of a continuous request missing a deadline, P_{md}, is an increasing function of the number of mini-cycles under all workloads. As illustrated in Figure 7, for $\lambda = 1.0$, N^*_{mc} should be equal to 4 so as to minimize $E[T_{nc}]$. However, at that point, $P_{md} \approx 0.016$, which violates the Prob$[\tau_{N_c} \geq T] \leq 0.01$ QoS requirement of continuous requests. To investigate this problem and the corresponding solution further, we perform the next experiment.

[18] Or, conversely, an optimal mini-cycle size.

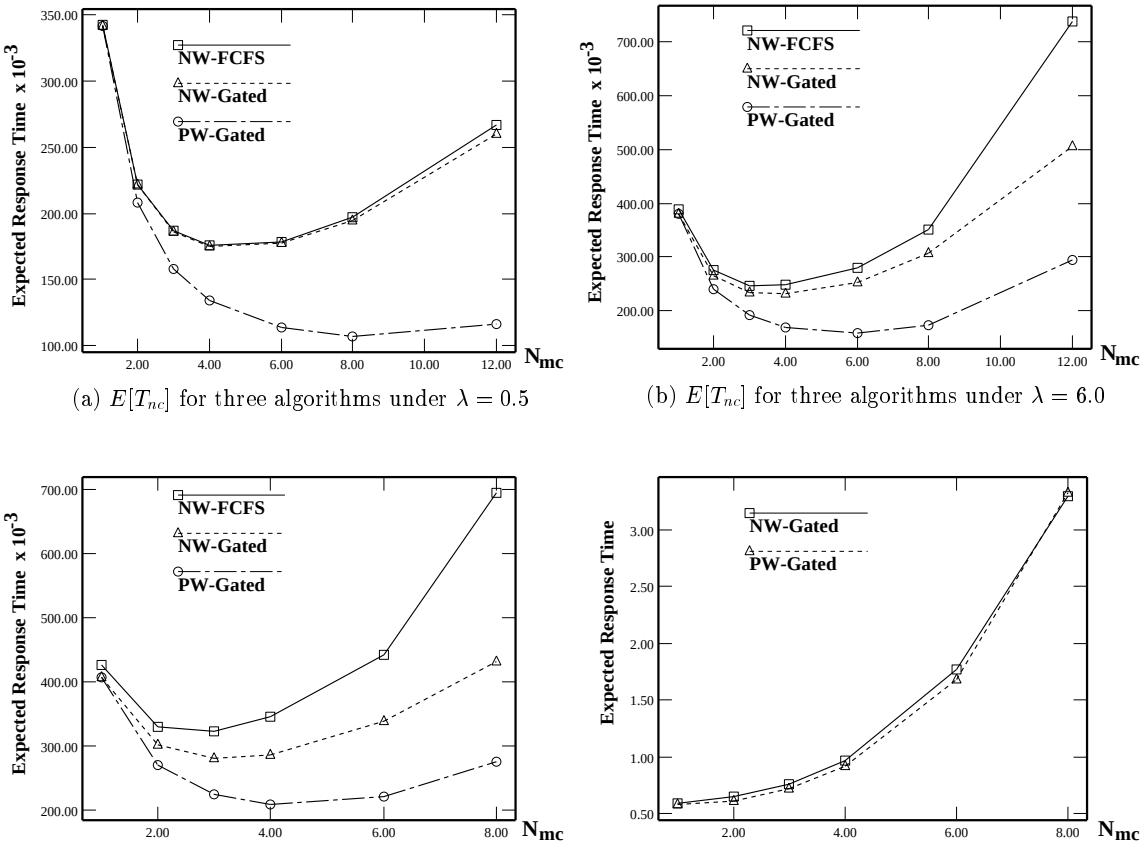
5.3 Experiment 3: Performance implications of optimal mini-cycle for various scheduling algorithms

In the previous experiment, we determined that there exists an optimal mini-cycle value, N^*_{mc}, at least for the NW-FCFS algorithm. In this experiment, we vary the number of mini-cycles in the Gated algorithms to determine whether a similar *convexity property* exists as well as to compare the performance (i.e., $E[T_{nc}], \sigma_{nc}, P_{md}$ and P_{ol}) of these algorithms. The result of this experiment is illustrated in Figure 9. Several important observations can be made from this figure. Firstly, a *convexity property* also exists in the Gated algorithms, which implies that there exists an optimal value for the number of mini-cycle, N^*_{mc}, here as well. Secondly, by comparing NW-FCFS and NW-Gated in Figures 9(a) and (b), we observe that under light loads most of the performance improvement in $E[T_{nc}]$ can be obtained by operating at a proper value of N_{mc}. That is, re-ordering the service of non-continuous requests in NW-Gated appears to result in only small performance gains, which is due to the fact that under light loads very few non-continuous requests are present in the non-continuous queue during the "gating instant". However, under all λ loadings, the PW-Gated algorithm improves the $E[T_{nc}]$ by a large margin because it both (a) *aggressively* services continuous requests when there are no non-continuous requests present in the system (i.e., it is work conserving in most mini-cycles) and (b) it services non-continuous requests (when they are present), in a *re-ordered* manner. That is, the "aggressive" service of continuous requests results in less "wasted" time during a cycle of length T as well as (on the average) in a greater accumulation of non-continuous requests in the queue at the "gating instant" which in turn results in greater seek optimization benefits. Under a moderate load (e.g., Figure 9(c)) NW-Gated and PW-Gated can reduce $E[T_{nc}]$ significantly because there are more non-continuous requests during the "gating instances", therefore, re-ordering of service pays off significantly. Lastly, under high loads (e.g., Figure 9(d)), NW-FCFS will not be able to support non-continuous requests with a reasonable response time, while NW-Gated and PW-Gated can still perform reasonably well. (Hence, NW-FCFS is not included on this graph.) That is, at such high loads NW-FCFS may not be stable and response time may grow to infinity, or as high as the maximum queue size will allow. Note that at a very high load (e.g., Figure 9(d)), NW-Gated and PW-Gated have comparable performance because the probability of not having any non-continuous requests to service in a mini-cycle (even without the "aggressive" service of PW-Gated) is very small at such loads.

In addition, Figure 10 depicts how the optimum number of mini-cycles changes as a function of load[19]. Observer that PW-Gated can sustain optimum performance at a greater number of mini-cycles and at higher loads, as compared to the non-work-conserving algorithms. This further illustrates that the (pseudo) work-conserving nature of this algorithm not only provides better performance to the non-continuous requests but also allows the system to be less "sensitive" to the seek optimization properties of the scheduling algorithm (and thus more flexible).

Figure 11 illustrates the variance of non-continuous requests under different arrival rates (λ)

[19] NW-FCFS does not cover the full range of loads, as it becomes unstable sooner than the Gated algorithms, as stated earlier.



(a) $E[T_{nc}]$ for three algorithms under $\lambda = 0.5$

(b) $E[T_{nc}]$ for three algorithms under $\lambda = 6.0$

(c) $E[T_{nc}]$ for three algorithms under $\lambda = 9.0$

(d) $E[T_{nc}]$ for NW-Gated, PW-Gated under $\lambda = 18.0$

Figure 9: $E[T_{nc}]$ for various algorithms under different arrival rates.

and different values for number of mini-cycles. We can conclude from these figures that the PW-Gated algorithm performs the best, both in terms of $E[T_{nc}]$ and in terms of σ_{nc}, and that the convexity property holds for all three algorithms. An important point to note is that the PW-Gated algorithm has a relatively flat curve for σ_{nc} under different workloads. This is an important characteristic since this implies that the PW-Gated algorithm not only reduces $E[T_{nc}]$ but also makes the response time of the non-continuous requests less variable, which should result in better quality-of-service to non-continuous customers.

Since PW-Gated performs much better than NW-Gated, from this point on we simply compare PW-Gated with NW-FCFS. Figure 12 illustrates that P_{md} for the PW-Gated algorithm is much lower than that for the NW-FCFS algorithm. The explanation here is that under different arrival rates of non-continuous requests, PW-Gated attempts to aggressively serve continuous requests whenever there are no non-continuous requests in the system. Therefore, it completes each mini-cycle as fast as possible, thereby increasing the probability of having a larger residual time in the last mini-cycle of each cycle. Since, according to our performance metric, a continuous request can only miss a deadline if it is processed after the end of a cycle of length T, PW-Gated reduces the probability of this undesirable event. Note that (in these experiments) with PW-Gated, it is possible for a system to operate at an optimal value of N^*_{mc} so as to minimize $E[T_{nc}]$ and at the

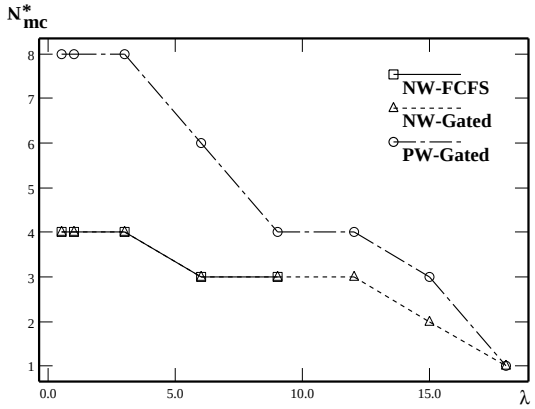
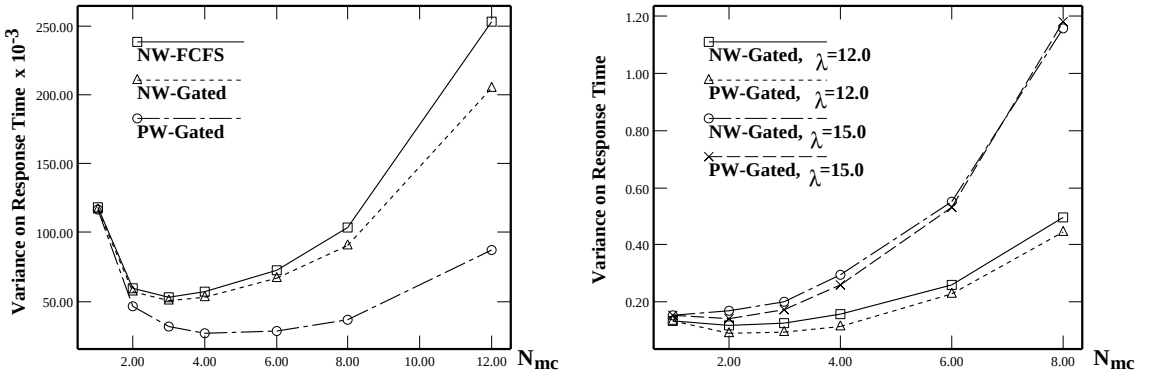


Figure 10: N^*_{mc} for various algorithms as a function of workload.

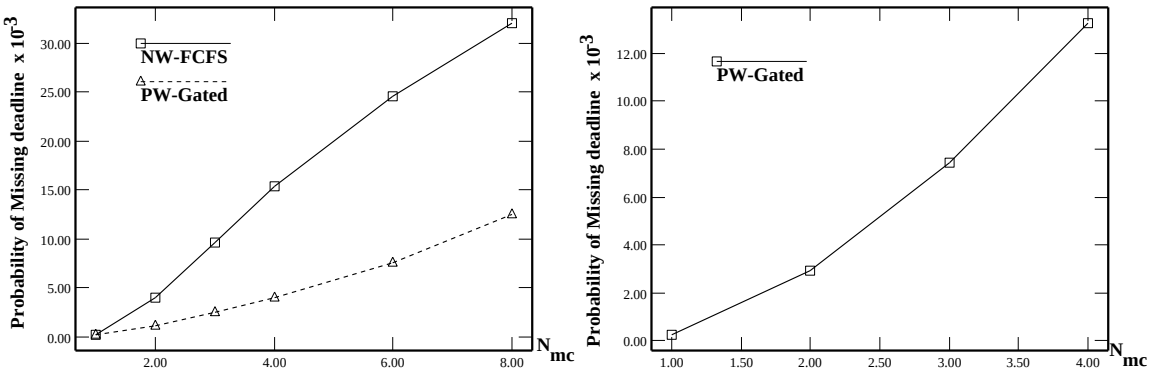


(a) σ_{nc} for three algorithms under $\lambda = 3.0$ (b) σ_{nc} for NW-Gated, PW-Gated under $\lambda = 12.0$ & 15.0

Figure 11: σ_{nc} for various algorithms under different arrival rates.

same time, satisfy the QoS requirement of the continuous requests. For example, at $\lambda = 9.0$, PW-Gated can operate at $N^*_{mc} = 4$ such that $E[T_{nc}]$ is minimized (refer to Figure 9(c)) and P_{md} is only equal to 0.004, which is much better than the (minimum) required QoS of Prob$[\tau_{N_c} \geq T] \leq 0.01$. From Figure 9, we observe that PM-Gated can reduce $E[T_{nc}]$, as compared to NW-FCFS, by as much as 60% while still satisfying the the QoS constraint for the continuous requests.

Lastly, we would like to point out that further improvements in response time of the non-continuous requests can be achieved through increases in the cycle time T. However, this improvement comes at the cost of increases in buffer space requirements for the continuous requests. Thus, in the next experiment we illustrate the tradeoff between improvements in response time for non-continuous requests, achieved through increases in the cycle time T, and the resulting increases in buffer space requirements for continuous requests.



(a) P_{md} for NW-FCFS and PW-Gate under $\lambda = 9.0$ (b) P_{md} for NW-FCFS and PW-Gate under $\lambda = 18.0$

Figure 12: P_{md} for NW-FCFS and PW-Gated under different arrival rates and mini-cycles.

5.4 Experiment 4: Resource and performance tradeoffs by increasing the scheduling period T

In the previous experiments, the cycle length T was fixed at 1.48754 seconds. We now increase T while keeping the number of continuous requests at $N_c = 24$. The rationale for performing these experiments is to determine whether we can further improve $E[T_{nc}]$ and P_{md} at the expense of larger buffer space requirements[20]. In these experiments, we increase the cycle length T by 25%, 50%, 75% and 100%. The mean transfer size of each continuous request is increased accordingly, i.e., so that the continuous requests can still maintain an ≈ 1.5 Mbps average display rate which corresponds to MPEG video streams[21]. Specifically, we have:

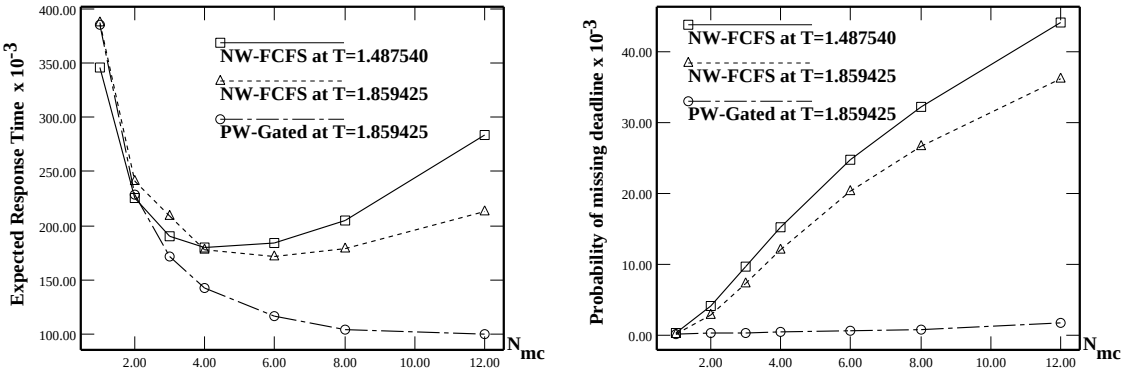
increase in T	length of T (in secs)	mean transfer size (Mbits)
T	1.487540	2.2500
1.25T	1.859425	2.8125
1.50T	2.231310	3.3750
1.75T	2.603195	3.9375
2T	2.975080	4.5000

Figure 13 illustrates the case where T is increased by 25%. To make a fair comparison, we consider the scheduling algorithms at their respective optimal operating points, N^*_{mc}, and observe that PW-Gated reduces $E[T_{nc}]$ by as much as 49.4%, as compared to NW-FCFS with $T = 1.48754$ (see Figure 13(c)), and at the same time still satisfies the QoS of continuous requests (see Figure 13(d)). However, this gain occurs at the cost of a (potential) 25% increase in buffer space requirements for the continuous requests.

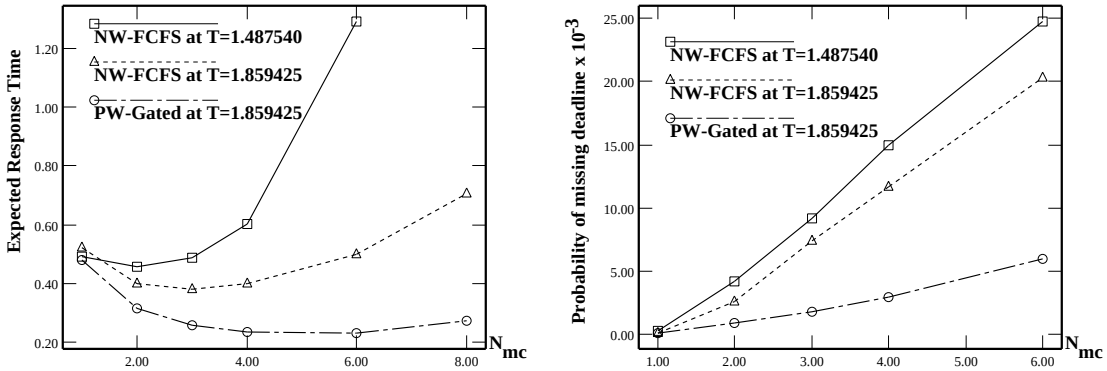
Figure 14 further illustrates improvements in $E[T_{nc}]$ for different values of T under PW-Gated. It shows that we can reduce $E[T_{nc}]$ by as much as 44% if we increase the value of T from 1.487540 to

[20] Recall that we can not transmit and thus must buffer data retrieved for all continuous requests in cycle i until the beginning of cycle $i + 1$.

[21] For example, if $T = 1.859425$, then we need to increase the average transfer size to 2.8125 Mbits such that with this cycle length T, on the average ≈ 1.5 Mbps are transfered.



(a) $E[T_{nc}]$ for NW-FCFS and PW-Gated under $\lambda = 1.0$ (b) P_{md} for NW-FCFS and PW-Gated under $\lambda = 1.0$



(c) $E[T_{nc}]$ for NW-FCFS and PW-Gated under $\lambda = 12.0$ (d) P_{md} for NW-FCFS and PW-Gated under $\lambda = 12.0$

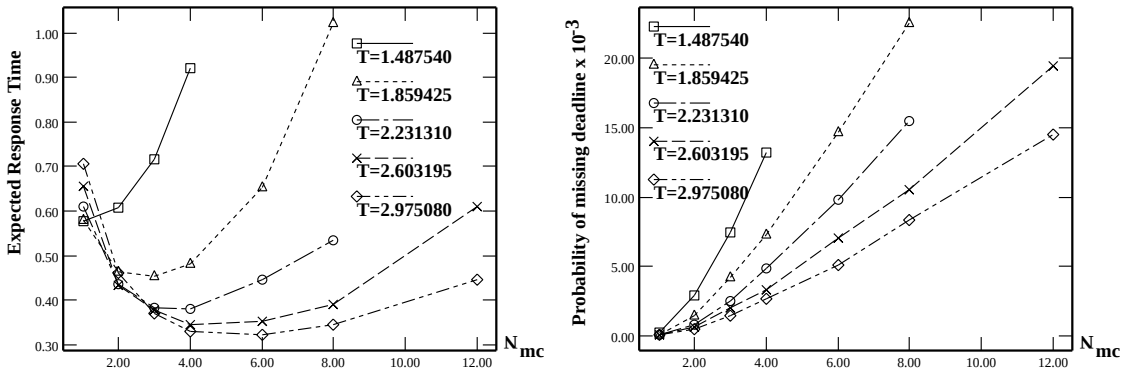
Figure 13: $E[T_{nc}]$ and P_{md} for NW-FCFS and PW-Gated under $1.25T$ and various loadings.

2.97508 (see Figure 14(a) with the corresponding probabilities of missing a deadline in Figure 14(b)). This improvement also, of course, comes at the cost of an increase in buffer space requirements of continuous requests.

To summarize, these experiments illustrate the following important points for mixed workload scheduling:

- The gated algorithms (e.g., NW-Gated and PW-Gated), as compare to the NW-FCFS algorithm, provide a much better performance for the non-continuous requests. Namely, the non-continuous requests will experience a lower expected response time, especially when the loading of the non-continuous requests is high.

- We can use the concept of mini-cycle to further improve the performance of the non-continuous requests. We also illustrate that there is a convexity property in the mini-cycle. That is, as we increase the number of mini-cycles, the performance of the non-continuous request will improve at first. This performance will degrade as we continue to increase the number of mini-cycles. The main reason is that by increasing the number of mini-cycle, the system can serve the non-continuous requests earlier. However, increasing the number of mini-cycles also implies that we are introducing more inefficiency in the system, that is, the aggregate seek



(a) $E[T_{nc}]$ for PW-Gated under $\lambda = 18.0$ (b) P_{md} for PW-Gated under $\lambda = 18.0$

Figure 14: $E[T_{nc}]$ and P_{md} for PW-Gated under different values of T and at high λ.

time overhead for serving the continuous requests will also increase.

- To provide the QoS guarantee to both the continuous and non-continuous requests, we have to consider both the probability of missing deadline (for the continuous request) and the expected response time (for the non-continuous requests). We illustrate how one can choose the optimal N_{mc} under different operating conditions, for example, arrival rate of non-continuous requests, scheduling interval,...etc.

- We also illustrate the resource tradeoffs of increasing the length of the scheduling period. In this case, we may be able to guarantee the QoS measure for the continuous requests and reduce the expected response time for the non-continuous requests at the expense of more system resource (e.g., buffer) and longer start-up latency.

6 Conclusions

In summary, the performance evaluation study of *mixed-workload* storage systems presented in this paper illustrated performance implications of important tradeoffs involving seek optimization and work-conserving nature of scheduling policies. This performance evaluation study exposed some important characteristics, e.g., the fact that maximum possible seek reduction does not always result in best system performance (in contrast to what is suggested in [16]). These important tradeoffs were exposed through the use of the mini-cycles technique. In this work, we consider two orthogonal approaches to scheduling of mixed workloads, namely (1) a technique (termed "mini-cycles") for reducing the waiting time of the non-continuous requests and 2) algorithms for providing better opportunities for seek optimization.

We investigated the performance tradeoffs under *high* continuous workloads. This is motivated by the cost-based consideration. We have shown that reasonable response time to non-continuous requests at *high* continuous workloads can be achieved clearly, *better* response time to non-continuous requests can be obtained at *lighter* continuous workloads.) We illustrate how to schedule the continuous request such that the QoS (e.g. probability of missing deadline) is bounded and at the same

time, the scheduling of the non-continuous requests so that the expected response time and the variance on the response time can be reduced. We show the convexity property of the mini-cycle technique and that there is an optimization problem to determine the optimal number of mini-cycle so as to service both the continuous and the non-continuous requests. Last but not least, we explore the possibility to reduce the $E[T_{nc}]$ further by increase the amount of buffer space required. The issues of an optimum mini-cycle as well as optimum buffer space cost/performance tradeoff are a topic of future work.

References

[1] S. Berson, S. Ghandeharizadeh, R. Muntz, , and X. Ju. Staggered Striping in Multimedia Information Systems. In *Internatinoal Conference on Management of Data*, pages 79–90, Minneapolis, Minnesota, May 1994.

[2] S. Berson, L. Golubchik, and R. R. Muntz. Fault Tolerant Design of Multimedia Servers. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 364–375, San Jose, CA, May 1995.

[3] E. Chang and A. Zakhor. Variable Bit Rate Mpeg Video Storage on Parallel Disk Arrays. In *Proceedings of SPIE Conference on Visual Communication and Image Processing*, pages 47–60, Chicago, Illinois, October, 1996.

[4] M. Chen, D. Kandlur, and P. Yu. Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Streams. *ACM Multimedia '93*, pages 235–242, 1993.

[5] A. E. Conway and N. D. Georganas. *Queueing Networks - Exact Computational Algorithms: A Unified Theory Based on Decomposition and Aggregation*. MIT Press, 1989.

[6] S. Ghandeharizadeh and R. R. Muntz. Design and implementation of scalable continuous media servers. *Parallel Computing Journal*, 24:91–122, 1998.

[7] L. Kleinrock. *Queueing Systems, Volume I*. Wiley-Interscience, 1975.

[8] S.W. Lau, John C.S. Lui, and L. Golubchik. Merging video streams: Complexity and heuristics. *Journal of Multimedia Systems*, 6(1):29–42, 1998.

[9] K.W. Law, J. C.-S. Lui, and L. Golubchik. Efficient support for interactive services in multi-resolution vod systems. *VLDB Journal*, 8(2):133–153, 1999.

[10] Mary Y.Y. Leung, John C.S. Lui, and L. Golubchik. Use of analytical performance models for system sizing and resource allocation in interactive video-on-demand systems employing data sharing techniques vod systems. *Accepted for publication in IEEE TKDE*, 2001.

[11] P. W. K. Lie, J. C.-S. Lui, and L. Golubchik. Threshold-based dynamic replication in large-scale video-on-demand systems. *Journal of Multimedia Tools and Applications*, 11(1), May, 2000.

[12] G. Nerjes, P. Muth, and G. Weikum. Stochastic Performance Guarantees for Mixed Workloads in a Multimedia Information System. In *Proc. of the IEEE Intl. Workshop on Research Issues in Data Engineering (RIDE'97)*, April 1997.

[13] G. Nerjes, P.Muth, M. Paterakis, Y. Romboyannakis, P. Triantafillou, and G. Weikum. Scheduling Strategies for Mixed Workloads in Multimedia Information Servers. *In Proc. of the IEEE Intl. Workshop on Research Issues in Data Engineering (RIDE'98)*, Feburary 23-24, 1998.

[14] Y. Romboyannakis, G. Nerjes, P. Muth, M. Paterakis, P. Triantafillou, and G. Weikum. Disk scheduling for mixed-media workloads in a multimedia server. In *Proc. of the ACM Multimedia Conference*, 1998.

[15] Chris Ruemmler and John Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer Magazine*, pages 17–28, March 1994.

[16] P. J. Shenoy and H. M. Vin. Cello: A disk scheduling framework for next generation operating systems. In *ACM Sigmetrics Conference*, June 1998.

[17] F. A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID - A Disk Array Management System For Video Files. *ACM Multimedia Conference*, pages 393–399, 1993.

[18] J. Wolf, H. Shachnai, and P. Yu. DASD Dancing: A Disk Load Balancing Optimization Scheme for Video-on-Demand Computer Systems. In *Proceedings of the ACM SIGMETRICS and Performance*, May 1995.

[19] P. S. Yu, M.-S. Chen, and D. D. Kandlur. Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management. *Third International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 44–55, November 1992.

[20] Z.-L. Zhang, J. Kurose, J. Salehi, and D. Towsley. Smooting, Statistical Multiplexing and Call Admission Control for Stored Video. *IEEE Journal on Selected Areas in Communication*, 15(16), August 1997.

Appendix: Formal Definition of the Queueing System

In this appendix we give a more formal definition of the behavior of the queueing system described in Section 4. Let L_{nc} be the queue length at queue Q_{nc}, and L_{nc}^{g} be the number of customers at queue Q_{nc} when they are gated (refer to Section 4). Finally, let *mc_count* be the number of the current mini-cycle, where $1 \leq mc\_count \leq N_{mc}$.

Before we proceed with the more formal definition of the queueing model, we first explain the notation used in this definition. Below, we describe the behavior of each queue as a collection of **event**'s and **msg_recv**'s, where an event can correspond to either an "internal" event (such as a service completion) or an "external" event (such as an arrival of a new customer). On the other hand, the receiving and sending of messages notation allows us to illustrate communication of control information (such as token passing) between the two queues. Each **event** and **msg_recv** is annotated with a **condition** and an **action**, meaning that the action is taken only if the condition is true. Each type of an event occurs at a given rate with a given distribution; for instance, the statement **event:** arrival(rate_type,$rate_{arrival}$), where rate_type is equal to "exponential" and $rate_{arrival} = \lambda$, means that the random variable corresponding to the time between consecutive events of type "arrival" is distributed exponentially with a mean of $1/\lambda$. Thus, we can view the description given below as an event-driven type of a description, i.e., each event is generated, independently of other events, with a given distribution and rate. Finally, in order to make the description of the model more general and at the same time more analytically tractable [7], the service of the continuous customer is described in terms of a series of stages; for instance, it could represent an r-stage Erlang distribution with $r = LASTSTAGE$ [7].

The initial settings of the following variables, used in the description below, are:

- $mc\_count = 1$: global variable which keeps track of the current mini-cycle number, i.e., $1 \leq mc\_count \leq N_{mc}$
- $done == false$: keeps track of whether the mini-cycle time has expired yet
- $stage == 1$: keeps track of in which stage of service the continuous customer is (as described above)
- $class == 1$: keeps track of to which class the continuous customer belongs (as described in Section 4 class switching represents going from one mini-cycle to the next)

The description of the behaviour of the queues is as follows.

Continuous Queue (NW-FCFS and NW-Gated)

1. **event:** change_stage(rate_type, $rate_{stage}$)
   /* changing of stage — service not finished */
   - **condition:** $(stage < LASTSTAGE)$ **and** $(token == 1)$
   - **action:** $stage++$

   /* changing of stage — service finished, timer not expired */

- **condition:** $(stage == LASTSTAGE)$ **and** $(token == 1)$ **and** $(done == false)$
- **action:**
  (a) $stage = 1$
  (b) $token = 0$
  (c) **if** $((class + 1) > N_{mc})$ **then** $class = 1$ **else** $class++$
  (d) **send-msg**(Q_{nc}, token_msg)

/* changing of stage — service finished, timer expired */

- **condition:** $(stage == LASTSTAGE)$ **and** $(token == 1)$ **and** $(done == true)$
- **action:**
  (a) $stage = 1$
  (b) $done = false$
  (c) **if** $((class + 1) > N_{mc})$ **then** $class = 1$ **else** $class++$

2. **event:** timeout(rate_type, $rate_{timeout}$)
   /* timer expired — still have the token */
   - **condition:** $(token == 1)$
   - **action:** $done = true$

   /* timer expired — don't have the token */
   - **condition:** $(token == 0)$
   - **action:**
     (a) $token = 1$
     (b) **send-msg**(Q_{nc}, timeout_msg)

Non-continuous Queue (NW-FCFS and NW-Gated)

1. **event:** serve(rate_type, $rate_{service}$)
   /* service non-continuous customer */
   - **condition:** $(L_{nc} > 0)$ **and** $(token == 1)$
   - **action:** $L_{nc}--$

2. **event:** arrival(rate_type, $rate_{arrival}$)
   /* arrival of a non-continuous customer */
   - **condition:** $true$
   - **action:** $L_{nc}++$

3. **msg_recv:** token_msg
   /* receive message of token being passed in */
   - **condition:** $true$
   - **action:** $token = 1$

4. **msg_recv:** timeout_msg

   /* receive message of timeout */

   - **condition:** $true$
   - **action:** $token = 0$

Continuous Queue (PW-Gated)

1. **event:** change_stage(rate_type, $rate_{stage}$)

   /* changing of stage — service not finished */

   - **condition:** $(stage < LASTSTAGE)$ **and** $(token == 1)$
   - **action:** $stage++$

   /* changing of stage — service finished, timer not expired */

   - **condition:** $(stage == LASTSTAGE)$ **and** $(token == 1)$ **and** $(done == false)$
   - **action:**
     (a) $stage = 1$
     (b) $token = 0$
     (c) **if** $((class + 1) > N_{mc})$ **then** $class = 1$ **else** $class++$
     (d) **send-msg**(Q_{nc}, token_msg)

   /* changing of stage — service finished, timer expired */

   - **condition:** $(stage == LASTSTAGE)$ **and** $(token == 1)$ **and** $(done == true)$
   - **action:**
     (a) $stage = 1$
     (b) $done = false$
     (c) **if** $((class + 1) > N_{mc})$ **then** $class = 1$ **else** $class++$

2. **event:** timeout(rate_type, $rate_{timeout}$)

   /* timer expired — still have the token */

   - **condition:** $(token == 1)$
   - **action:**
     (a) $done = true$
     (b) **if** $((mc\_count + 1) > N_{mc})$ **then** $mc\_count = 1$ **else** $mc\_count++$

   /* timer expired — don't have the token */

   - **condition:** $(token == 0)$
   - **action:**
     (a) $token = 1$
     (b) **if** $((mc\_count + 1) > N_{mc})$ **then** $mc\_count = 1$ **else** $mc\_count++$
     (c) **send-msg**(Q_{nc}, timeout_msg)

3. **msg_recv:** token_msg
   /* receive message of token being passed in */
   - **condition:** $true$
   - **action:** $token = 1$

Non-continuous Queue (PW-Gated)

1. **event:** serve(rate_type, $rate_{service}$)
   /* service non-continuous customer */
   - **condition:** $(L^g_{nc} > 0)$ **and** $(L_{nc} > 0)$ **and** $(token == 1)$
   - **action:**
     (a) $L^g_{nc} = L_{nc}$
     (b) $L_{nc} = 0$

2. **event:** serve(rate_type, $rate_{service}$)
   /* service last non-continuous customer */
   - **condition:** $(L_{nc} == 0)$ **and** $(token == 1)$ **and** $(mc\_count < N_{mc})$
   - **action:**
     (a) $L^g_{nc} = 0$
     (b) **send-msg**(Q_c, token_msg)

3. **event:** serve(rate_type, $rate_{service}$)
   /* service last non-continuous customer */
   - **condition:** $(L_{nc} == 0)$ **and** $(token == 1)$ **and** $(mc\_count == N_{mc})$
   - **action:** $L^g_{nc} = 0$

4. **event:** arrival(rate_type, $rate_{arrival}$)
   /* arrival of a non-continuous customer */
   - **condition:** $true$
   - **action:** $L_{nc}++$

5. **msg_recv:** token_msg
   /* receive message of token being passed in */
   - **condition:** $L_{nc} > 0$
   - **action:**
     (a) $token = 1$
     (b) $L^g_{nc} = L_{nc}$
     (c) $L_{nc} = 0$
   - **condition:** $L_{nc} == 0$
   - **action:** **send-msg**(Q_c, token_msg)

6. **msg_recv:** timeout_msg

   /* receive message of timeout */

   - **condition:** $true$
   - **action:**

     (a) $token = 0$

     (b) $L_{nc} = L_{nc} + L^g_{nc}$

     (c) $L^g_{nc} = 0$