Analytical Models for Mixed Workload Multimedia Storage Servers

Edmundo de Souza e Silva [a,1] H. Richard Gail [b]
Leana Golubchik [c,2] John C.S. Lui [d,3]

[a] *Federal University of Rio de Janeiro, edmundo@nce.ufrj.br*

[b] *IBM T.J. Watson Research Center, rgail@us.ibm.com*

[c] *Department of Computer Science & UMIACS, University of Maryland, leana@cs.umd.edu*

[d] *Department of Computer Science & Engineering, The Chinese University of Hong Kong, cslui@cse.cuhk.edu.hk*

Abstract

Efficient resource management is essential in providing *scalability* in large multimedia information systems serving a variety of applications. In order to explore design tradeoffs in such systems, what is needed are accurate *analytical* models of *mixed* workload servers with *tractable* solutions. Motivated by the need for mixed workload storage servers as well as the need for performance studies of such systems, we first present a family of scheduling algorithm for mixed workload storage servers. We then develop a set of corresponding *analytical non-Markovian* models. We show how our analytical solution methodology applies to this entire family of models and how it can be used to obtain performance measures of interest for different classes of workloads. Lastly, we illustrate through numerical examples how such models can be used to study performance tradeoffs and facilitate the making of design choices in mixed workload storage servers.

Key words: embedded Markov chains, mixed workload scheduling, multimedia storage servers, uniformization.

[1] This research was partially supported by CNPq grants: Protem-CC and Pronex.
[2] This research was supported in part by the NSF CAREER grant CCR-98-96232.
[3] This research was supported in part by the RGC and CUHK Mainline Research Grant.

1 Introduction

Many modern applications can benefit cost-wise from sharing resources such as network bandwidth and disk bandwidth. In addition, it is desirable for information systems 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, 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 is to simply share the resources among the different classes of requests with a *best-effort* attempt to meet the performance or QoS requirements of each. Another approach is to *partition* the available resources between the different classes of workloads, i.e., to essentially maintain separate and independent servers. However, resource partitioning is, in general, 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, in addition, incur a penalty for having to maintain consistency between multiple copies of the data. Thus, a more sensible approach is 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 order to explore such designs as well as facilitate efficient studies of tradeoffs of the possible design approaches, what is needed are accurate *analytical* models of mixed workload multimedia systems with *tractable* solutions.

Motivated by such applications, in this paper we consider analytical models of multimedia storage servers which, in general, can serve a variety of applications, requesting video, image, audio, and text data. We focus on the storage system and assume that the network can deliver the necessary performance. We consider two classes of workloads: (1) continuous (or real-time), and (2) non-continuous (or non-real-time). For instance, the real-time workload with continuity-type requirements may correspond to requests for video streams, whereas the non-real-time workload may correspond to billing inquiries about the videos, requests for thumbnail images corresponding to particular scenes in a video, and so on. In the remainder of the paper, we will use the terms "real-time" and "continuous" interchangeably; likewise for the terms "non-real-time" and "non-continuous". Clearly, the two types of workloads have different performance and QoS requirements. For instance, the real-time workload requirements might include delivery of data at a particular rate (e.g., at 4 Mbps for an MPEG-2 stream) with little jitter, whereas the non-real-time workload requirements might include short response time.

There is a large body of work on the design of continuous media servers, which include [1,20,23] (see [10] for a detailed survey of the literature). This work focuses mostly on data layout and retrieval and delivery techniques which facilitate the maintaining of *continuity* in data delivery while providing either deterministic or statistical QoS guarantees. The scheduling of *mixed* workloads has not received as much attention. In [17,18,16,19], the authors discuss such scheduling techniques as well as present fairly *coarse* analytical models of the system. We build on this work in [11], in studying the tradeoffs involved in serving mixed workloads on the same storage server. Hierarchical scheduling with a corresponding taxonomy of schemes is considered in [21].

All these studies illustrate one important point — there are many tradeoffs in designing mixed workload multimedia storage servers which often correspond to "non-obvious" design choices. For instance, the main tradeoff we explore in [11] 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. The performance consequences arising from these and other tradeoffs are often difficult to assess, and their evaluation can lead to significantly different designs of the system. Such design choices are better studied through *analytical* models, as it is desirable to obtain results quickly, especially at design time. The potentially higher accuracy that can be obtained through either detailed simulations or measurements is often not needed at design time or simply not possible or economical to achieve.

Motivated by the need for mixed workload storage servers as well as the need for performance studies that can lead to better designs of such servers, we first present a family of scheduling algorithm for mixed workload storage servers. We then develop a family of *analytical* non-Markovian models, corresponding to these scheduling algorithms, with *tractable* analytical solutions. We show how our solution methodology applies to this entire family of models and how it can be used to obtain performance measures of interest for both classes of workloads. Finally, we illustrate, through numerical examples, how such models can be used to study performance tradeoffs and facilitate making of design choices in mixed workload storage servers.

2 Scheduling

In this section, we first review the basic concept of cycle-based (or group-based) scheduling [4,23,24], which is commonly used for serving *continuous* media requests in storage servers. We then describe how *statistical* quality-of-service (QoS) provisions are made for continuous requests. Finally, we present a family of scheduling algorithms for mixed workload servers which we model and analyze in Sections 3 and 4.

2.1 Cycle-Based Scheduling

In cycle-based scheduling algorithms, the retrieval of data from the disk subsystem for serving continuous requests is performed on a cyclic basis, where in each cycle is of length T the system retrieves data for N_c continuous requests. Here we assume that the viewing clients have relatively little buffer space and thus the server is responsible for maintaining the continuity in data delivery. Consequently, 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. If the data delivery is not offset by one cycle from data retrieval, then jitter may occur due to seek optimization, as explained below. The use of cycle-based scheduling is motivated by the increased opportunities for performing seek optimization, i.e., data blocks needed for service of a group of continuous requests are retrieved using a SCAN-type algorithm. Since the transmission of data is offset by one cycle from its retrieval, the *order* of data retrieval within a cycle does not affect the jitter characteristics of the transmission; hence, the SCAN algorithm.

The cost of this optimization is that the system may need additional buffer space to hold the retrieved data until the beginning of the next cycle. This cycle-based or group-based approach to serving continuous streams is, for instance, suggested in [4,23,24], and the tradeoff between improved utilization of the disk bandwidth (due to seek optimization) and the need for additional buffer space is analyzed, for instance, in [2,4,24]. Note that, in general, larger values of N_c afford better seek optimization opportunities; however, they may also result in larger buffer space requirements.

2.2 Computing Cycle Time

One important design parameter in cycle-based scheduling is 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 served by the system within a cycle and the degree of QoS that the system can provide. Such QoS guarantees can be provided by the system either deterministically (by considering worst case scenarios) or stochastically. The undesirable effect of deterministic QoS provisions is that they can result in poor disk bandwidth utilization, e.g., when the video streams have variable bit rate characteristics and there is a large deviation between the peak and the mean rates. Thus, in the remainder of the paper we consider *statistical* QoS guarantees.

Let τ_{N_c} be the random variable representing the service time of N_c continuous requests. Then

$$\tau_{N_c} = \tau_{seek}(N_c) + N_c * (\tau_{rot} + \tau_{tfr}), \quad (1)$$

where $\tau_{seek}(N_c)$ is the random variable corresponding to the total seek time of N_c requests incurred in one scan of a disk. As explained below, we will assume this seek time to be deterministic, *given* N_c. Random variables τ_{rot} and τ_{tfr} correspond to the rotational latency for each of the N_c requests and to the transfer time of each of the N_c requests, respectively.

The system guarantees that the probability of the event that τ_{N_c} is greater than T is less than some predefined system parameter, p. That is

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

One can use, for instance, the Chernoff bound [14] (e.g., as in [16,25]) to determine the value of T so as to serve 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 τ_{rot} and τ_{tfr}, 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 [10], we have

$$F^*_{N_c}(s) = e^{-s\ \tau^{max}_{seek}(N_c)} \left[F^*_{rot}(s) F^*_{tfr}(s)\right]^{N_c}. \quad (3)$$

Here we assume a worst case (deterministic) seek of the disk scan as a function of N_c, which is represented by $\tau^{max}_{seek}(N_c)$, hence the first term in Equation (3).

Let $M_{N_c}(s)$ be the moment generating function for the random variable τ_{N_c}. Since $M_{N_c}(s) = F^*_{N_c}(-s)$, applying the Chernoff bound gives [15]

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

Using standard numerical solution techniques, the optimal θ^* which gives the tightest upper bound can be obtained, which then yields 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 overflow situations. For example, the system can allow an "overrun" of data retrieval into the next cycle, i.e., finish serving 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. Alternatively, the system can stop serving the requests in the overflowing cycle i and proceed to serve the next N_c continuous requests in cycle $i+1$. (A more general scheme allows the overruns to "terminate" after some predefined number of cycles,

L.) Our methodology can be used to model any one of these approaches to handling overflow.

2.3 A Family of Mixed Workload Scheduling Algorithms

In what follows, we consider a system which *always* has N_c continuous requests present. That is, our interest is in the performance of the system under *high* continuous requests loads, partly because 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 real-time requests, provided that QoS requirements can be satisfied for both classes of customers. Thus, *high* real time workloads correspond to reasonable and important operating points at which to consider our system. The family of scheduling algorithms given below explore several degrees of freedom in serving the two classes of workloads, including: (1) ordering of service of non-continuous requests as well as (2) the work conserving nature (or lack thereof) of the system, or in other words, "greediness" in serving continuous requests.

Non-Greedy FCFS (NG-FCFS)
We have already described how the system schedules the retrieval of continuous requests using cycle-based scheduling and scanning of the disk. Thus, whatever time remains in the cycle can be used to serve any non-continuous request present in the system. We first consider serving these in a FCFS manner, and call this algorithm *Non-Greedy FCFS* (NG-FCFS). More formally, assume there are two classes of requests in our system, class C (for continuous requests) and class NC for (non-continuous requests). Let N_c be the (fixed) number of class C requests which is used to compute the cycle time T, as in Equation (4) above. Thus, in the NG-FCFS algorithm, we first serve N_c customers within a single period of length T. If after serving all N_c customers the system still has some residual time within the period, that remaining time is dedicated to serving non-continuous requests in a FCFS manner. If there is not sufficient time remaining in T to serve a non-continuous request, then no additional requests are served 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 server will not schedule the waiting continuous requests but instead, will remain idle until the end of the period. The necessity to be non-work conserving is motivated by the need to maintain a specific rate of data delivery for continuous requests. Any "early" data retrieval (i.e., earlier than is dictated by the desired delivery rate) will result in increasing growth in buffer space requirements, either at the storage server or at the client, depending on the system architecture. An example of the NG-FCFS scheduling algorithm is illustrated in Figure 1,

where $N_c = 5$ and an overflow event is depicted in the second period.

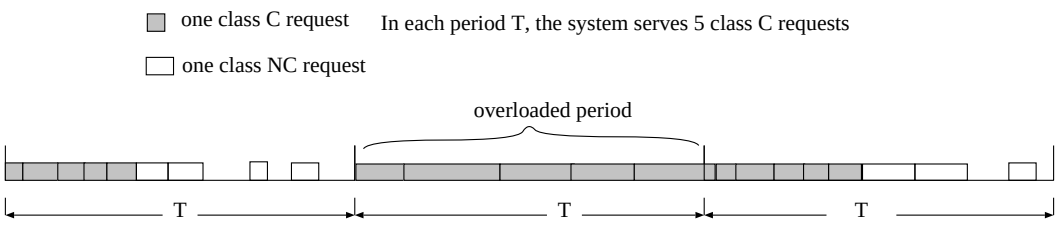

Fig. 1. Non-Greedy FCFS Algorithm: Within a cycle, serve class C first, then serve class NC in FCFS manner.

Although NG-FCFS algorithm provides reasonable performance characteristics for the continuous customers, it can result in long waiting times for the non-continuous requests. One approach to improving the response time of non-continuous requests is to serve them in groups instead of in a FCFS manner, i.e., provide similar seek optimization opportunities as for continuous requests by serving groups of non-continuous requests in a SCAN-type manner. This approach is taken in the next algorithm.

Non-Greedy gated (NG-Gated)
The *Non-Greedy Gated* (NG-Gated) algorithm is similar to the NG-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 minimization, the system can re-order 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 cycle, since new requests that are closer to the head's position would have a higher priority of service. To alleviate this problem, we use the following *gated* discipline. When the system completes service of class C requests before the end of a cycle, the system switches to serving 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 previous 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 gives priority to old NC requests over new ones while trying to make efficient use of the disk; thus, some notion of *fairness* is also provided. Similar to the NG-FCFS algorithm, a non-continuous request is not taken into service if there is not sufficient time to finish serving that request before the end of the cycle. This algorithm is non-work-conserving, because it is possible that the server is idle while there are some class C requests waiting for service (e.g., when there are no non-continuous requests present in the system and there is still residual time in a cycle).

The NG-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. 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.

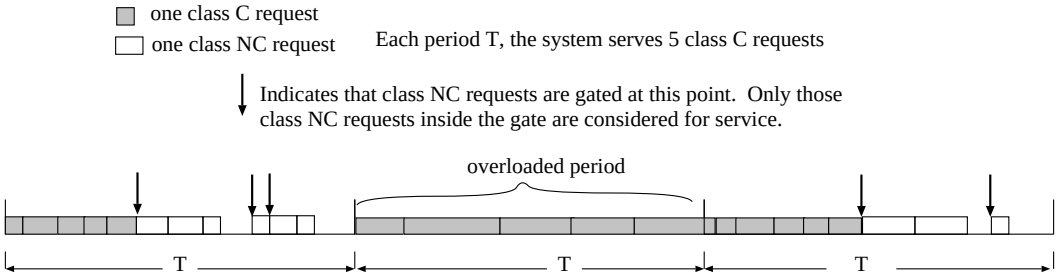

Fig. 2. NG-Gated Algorithm: Within a cycle, serve class C, then gated service for class NC.

Even under the gated service discipline, 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. Only then, if there is some residual time in the cycle, the service of the non-continuous request can begin.

To improve the response time of non-continuous requests and, at the same time, not significantly degrade the performance characteristics of the continuous requests, we can generalize these scheduling algorithms as follows. We divide the cycle of length T into N_{mc} mini-cycles, each of length T/N_{mc}. Within each mini-cycle, the system uses the NG-FCFS (or the NG-Gated) algorithm to serve N_c/N_{mc} continuous requests. In practice, of course, 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. 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 the presence of two opposing effects.

(1) By serving all continuous requests within *one* scan, we achieve better disk bandwidth utilization. In this case, greater seek optimization opportunities exist for the continuous requests, which can result in *more* time available in a cycle for serving *non-continuous* requests.
(2) By serving the continuous requests in *many* mini-cycles, we reduce the opportunities for seek optimization of the continuous requests, and thus we reduce the amount of cycle time that could have been used for service of non-continuous requests. However, we potentially improve the response time of the non-continuous requests by serving some of them earlier in a cycle. In addition, it is also possible that some continuous requests, especially those that are served in the later mini-cycles, will have a lower

probability of completing service before the end of a cycle.

These tradeoffs give rise to the following optimization problem. How does one find an *optimal* number of mini-cycles so as to minimize the expected response time of the non-continuous requests while providing the required QoS to continuous requests. An analytical model can be used to explore these tradeoffs.

Clearly, one problem with the non-greedy algorithms is that idle times may be *wasted* at the end of *each mini-cycle*. To improve on these algorithms, we describe another class, which we term *greedy* algorithms. The greedy algorithms can exploit both FCFS and Gated type service of non-continuous requests. In the interests of brevity, below we only describe the gated version; the modifications needed for the FCFS version are straightforward.

Greedy Gated (G-Gated)
Given N_{mc} mini-cycles, in the first mini-cycle the system begins by serving 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 service of the next N_c/N_{mc} class C requests. If there are class NC requests waiting, the system will serve these class NC requests in a gated fashion, as in the NG-Gated algorithm above. As in other algorithms, if there is not sufficient time remaining in T to serve a non-continuous request, then the server remains idle until the end of that cycle. Thus, the server switches back to serving class C requests when either: (1) there are no more class NC requests in the queue or (2) the mini-cycle ends *and* the system has *not* managed to get "ahead" on service of class C requests during some previous mini-cycle. For instance, if during the first mini-cycle the system manages to serve $2\frac{N_c}{N_{mc}}$ class C requests, then it will continue to serve class NC requests at the beginning of the second mini-cycle, since the system is "ahead" on service of class C requests in this example. The system behavior continues in this manner for the first $N_{mc} - 1$ mini-cycles. For the last mini-cycle, the system will behave as in the NG-Gated algorithm.

This algorithm is greedy because, during the first $N_{mc} - 1$ mini-cycles, the server is never idle. Only in the last mini-cycle is there a possibility of the server being idle while there are continuous requests in the system. Another important point to observe is that the greedy service of class C requests in the first $N_{mc} - 1$ mini-cycles should result in a lower probability of overflow for class C requests that belong to the later mini-cycles (i.e., those served towards the end of the cycle), thereby reducing jitter in data delivery. Figure 3 depicts the G-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 batch of 5 class C requests, and as soon as there are no class NC requests in the system, it immediately switches to the second batch of 5 class C requests (even though

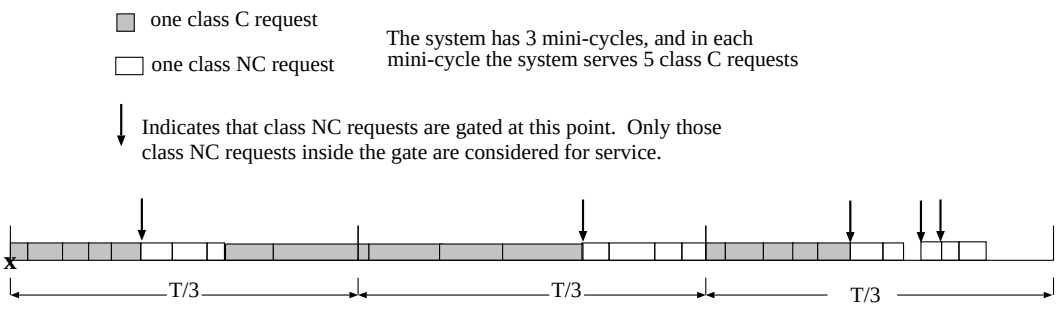

Fig. 3. G-Gated Algorithm: Serve class C requests, then gated service for class NC requests.

the first mini-cycle has not yet expired). In the last mini-cycle, the system uses the NG-Gated algorithm to serve class NC customers.

In some sense, the above algorithm is only partially greedy, because it takes into consideration the notion of being "ahead" on service of class C requests. We can imagine an even more greedy algorithm, i.e., one that *always* schedules service of another $\frac{N_c}{N_{mc}}$ class C requests whenever a mini-cycle ends. We will distinguish between these two approaches by referring to the former one as a *partially greedy* algorithm and to the latter one as a *greedy* algorithm. Lastly, it is easy to observe that when the number of mini-cycles is equal to 1, the partially greedy and the greedy algorithms behave just like the non-greedy algorithm. In summary, we have discussed six algorithms, i.e., Non-greedy FCFS, Non-greedy Gated, Partially Greedy FCFS, Partially Greedy Gated, Greedy FCFS, and Greedy Gated. Clearly, other algorithms are possible, but these six are sufficient to illustrate the range of possibilities as well as the usefulness of our models and the corresponding solution technique.

3 Analytical Models

In this section, we present a family of analytical performance models of the scheduling algorithms described in Section 2. Note that in the mixed workload scheduling algorithms deterministic events are present; for instance, the end of a mini-cycle is a deterministic event. Thus, in order to model these algorithms accurately, we need to consider analytical models that are *non-Markovian* in nature. In order to analyze these models, we adapt the methodology developed in [8] (see also related work in [5,7,9]). This approach is based on the use of embedded Markov chains to analyze *non-Markovian* stochastic processes and involves three fundamental steps: (1) identifying a sequence of embedded points, (2) deriving transition probabilities from one embedded point to another, and (3) evaluating performance measures of interest based on the steady state probabilities and appropriate reward functions at these embedded points.

In the models developed here, the embedded points we consider occur at the start (or end) of each mini-cycle. We model the service of a set of class C customers, which are served in a *single* mini-cycle, using an Erlang distribution with K stages (as explained in Section 4), where each stage has an exponential parameter α. The arrival process of class NC customers is Poisson with a rate λ. We model the service of one NC customer using an exponential distribution, the rate of which is a function of the service discipline used to serve this class of customers, as explained below. Finally, we assume a finite buffer size for class NC customers, with a maximum buffer size of B.

Events of interest that occur in our models are either exponential or deterministic. The deterministic event corresponds to the duration of a mini-cycle, and it occurs every T/N_{mc} time units. The exponential events are of three types: (1) the service completion of one Erlang stage corresponding to part of the work associated with the class C customers served in a mini-cycle; (2) the arrival of a class NC customer; and (3) the departure of a class NC customer after its service is completed. The rates of these exponential events are a function of the type of a mixed scheduling algorithm under consideration (see Section 4 for details).

Recall that the mixed workload scheduling algorithms are divided into two classes, namely, the non-greedy (NG) and the greedy (G). We will consider the partially greedy variation later in this section; however, it is similar in spirit to the greedy. Each algorithm has two variants according to the service discipline of the class NC requests, namely, first come first served (FCFS) and gated (G). In this section, we present the variation of the models for which overruns of class C customers to the next period are not allowed, i.e., stages of class C customers still present in the system at the end of the period of length T are dropped. In [6] we give details of how these models can be extended to include the two approaches that allow the handling of overruns, as suggested in Section 2.

3.1 Non-Greedy Algorithms

At the beginning of a mini-cycle, K stages of class C requests are introduced into the system (the determination of K is given in Section 4). If the service of all K stages is completed before the end of the current mini-cycle, then class NC requests are served. If no class NC customers are present, the server remains idle until the end of the mini-cycle. In the NG-FCFS model, the class NC requests are served exhaustively, while in the NG-Gated model, they are served in a gated fashion. In the exhaustive case, all class NC requests have an exponential service requirement of rate β, while in the gated case these requests are served at a rate that depends on the number of class NC requests

present when the gate is closed. If there are g class NC requests at that time, then the corresponding exponential service rate is β_g.

3.1.1 Model for the Non-Greedy FCFS Algorithm

The states of the NG-FCFS model are of the form (i, j), where i represents the number of class C stages in the system and j represents the number of class NC requests in the system. Since the buffer size for class NC requests is limited to B, $j \in \{0, \ldots, B\}$. Since no overruns of class C stages are allowed and since K stages are entered into the system at each of the N_{mc} mini-cycles, $i \in \{0, \ldots, N_{mc}K\}$. The events of interest with corresponding transition rates include:
(1) exponential event for the departure of a class C stage:

$$(i, j) \xrightarrow{\alpha} (i-1, j), \qquad (i > 0) \tag{5}$$

(2) exponential event for the arrival of a class NC request:

$$(i, j) \xrightarrow{\lambda} (i, j+1), \qquad (j < B) \tag{6}$$

(3) exponential event for the departure of a class NC request:

$$(0, j) \xrightarrow{\beta} (0, j-1), \qquad (j > 0) \tag{7}$$

(4) deterministic event corresponding to the end of one of the first $N_{mc} - 1$ mini-cycles in a period of length T:

$$(i, j) \longrightarrow (i + K, j), \tag{8}$$

This corresponds to adding K additional stages of work for class C requests at the beginning of the next mini-cycle.
(5) deterministic event corresponding to the end of the last mini-cycle in a period of length T:

$$(i, j) \longrightarrow (K, j), \tag{9}$$

Note that overflow occurs when the system is in state (i, j) where $i > 0$ at the end of a period of length T. Since in the model presented above we assume that overruns are not allowed, the C stages remaining at the end of the period are dropped and the system starts the next period of length T with K stages of work for class C requests (and all class NC requests present at that time).

3.1.2 Model for the Non-Greedy Gated Algorithm

The model of the NG-Gated algorithm is slightly more complicated. Specifically, in the state space description corresponding to class NC requests, we need to account for: (a) the total number of class NC requests in the system, (b) the number of class NC requests that were present when the gate closed *and* are still in the system, and (c) the number of class NC requests that were present when the gate closed, since the service rate of a class NC request depends on that number (recall we are using a SCAN-type algorithm to serve class NC requests in this case). Thus, the states of the NG-Gated model are of the form (i, j, h, g), where i represents the number of class C stages in the system, j represents the *total* number of class NC requests in the system, h represents the number of class NC requests in the system which were present at closing of the gate *and* are *still* in the system, and g indicates the number of class NC requests that were present when the gate last closed (the value of g dictates the rate at which class NC requests are currently being served). Note that $h \leq j$, since h counts departures and j counts both arrivals and departures of class NC requests. The value of h can only decrease until it reaches zero; when it becomes zero before the end of a mini-cycle, a new gate closes and a new set of class NC requests begin service. Furthermore, $h \leq g$, since initially their values are identical at the instant that the gate closes, but h decreases due to departures of class NC requests while g remains constant. This condition holds until a new gate closes or the mini-cycle ends. Thus in the gated case, $i \in \{0, \ldots, N_{mc}K\}$ and $j, h, g \in \{0, \ldots, B\}$. The events of interest for this model include:

(1) exponential event for the departure of a class C stage:

$$(i, j, 0, 0) \xrightarrow{\alpha} (i-1, j, 0, 0), \qquad (i > 1) \tag{10}$$

$$(1, j, 0, 0) \xrightarrow{\alpha} \quad (0, j, j, j), \tag{11}$$

where Equation (11) represents switching service from class C requests to class NC requests. At that instant the gate is closed, and j class NC requests are eligible for service.

(2) exponential event for the arrival of a class NC request:

$$(i, j, h, g) \xrightarrow{\lambda} (i, j+1, h, g), \qquad (j < B) \tag{12}$$

(3) exponential event for the departure of a class NC request:

$$(0, j, h, g) \xrightarrow{\beta_g} \quad (0, j-1, h-1, g), \qquad (h > 1) \tag{13}$$

$$(0, j, 1, g) \xrightarrow{\beta_g} (0, j-1, j-1, j-1), \tag{14}$$

where Equation (14) represents the event that the gate is again closed and there are $j-1$ class NC requests eligible for service.
(4) deterministic event corresponding to the end of one of the first $N_{mc}-1$ mini-cycles in a period of length T:

$$(i,j,0,0) \longrightarrow (i+K,j,0,0), \qquad (i>0) \tag{15}$$
$$(0,j,h,g) \longrightarrow \quad (K,j,0,0), \tag{16}$$

In all cases, K additional stages of work from class C requests are scheduled at the beginning of the next mini-cycle. Equation (15) corresponds to serving class C requests both before and after the start of the mini-cycle, while Equation (16) corresponds to switching service from class NC to class C when a new mini-cycle begins.
(5) deterministic event corresponding to the end of the last mini-cycle in a period of length T:

$$(0,j,h,g) \longrightarrow (K,j,0,0), \tag{17}$$
$$(i,j,0,0) \longrightarrow (K,j,0,0), \qquad (i>0) \tag{18}$$

Equation (18) corresponds to the overflow condition, i.e., $i>0$ stages of class C requests are dropped before starting the next time period of length T.

3.2 Greedy Algorithms

We now consider a greedy model, for which K stages of work corresponding to class C requests can be introduced into the system whenever the number of class NC requests becomes zero before the end of a mini-cycle. We consider two different variants depending on how the introduction of the K stages of class C into the system at the beginning of a mini-cycle is done. The first variant is the *greedy* strategy, wherein these K stages are always introduced into the system at the beginning of a mini-cycle. The second is the *partially greedy* strategy wherein these K stages are not introduced into the system *if* the system is "ahead" of schedule on the service of class C requests (see Section 2 for details of the corresponding scheduling algorithms). Thus, in the partially greedy case, at the *beginning* of the k^{th} mini-cycle, where $k \in \{2,3,\ldots,N_{mc}\}$, K additional stages of class C work will *not* be introduced if the server has already served an amount of class C work corresponding to the n^{th} mini-cycle where $n \geq k$. The motivation for the partially greedy strategy is to allow class NC requests to receive service earlier. In both variants, the system only introduces at most $N_{mc} * K$ stages of work for class C requests in a period of length T. Similar to the non-greedy case, class NC requests can be served based on the FCFS service discipline or based on the gated service discipline.

As before, the service requirement of a class NC request is exponential with a rate depending on the service discipline (i.e., the rate is β for exhaustive service and β_g for gated service when g customers are present at the closing of the gate).

3.2.1 Model for the Greedy FCFS Algorithm

In the case of the G-FCFS algorithm, the states of the model are (i, j, w), where i represents the number of class C stages in the system, j represents the number of class NC requests in the system, and w represents the number of class C "chunks of work" (i.e., units of K stages) that have been introduced into the system. If k represents the current mini-cycle where $k = 1, \ldots, N_{mc}$, then $w \geq k$, since class C stages can be introduced before the end of a mini-cycle, as well as at the start of a new mini-cycle. These state variables range over $i \in \{0, \ldots, N_{mc}K\}$, $j \in \{0, \ldots, B\}$, and $w \in \{1, \ldots, N_{mc}\}$. The possible events of interest include:
(1) exponential event for the departure of a stage of a class C request:

$$(i, j, w) \xrightarrow{\alpha} (i-1, j, w), \qquad (i > 1) \tag{19}$$
$$(1, j, w) \xrightarrow{\alpha} (0, j, w), \qquad (j > 0) \tag{20}$$
$$(1, 0, w) \xrightarrow{\alpha} (K, 0, w+1), \qquad (w < N_{mc}) \tag{21}$$
$$(1, 0, N_{mc}) \xrightarrow{\alpha} (0, 0, N_{mc}), \tag{22}$$

Equation (21) indicates the greedy characteristic of the algorithm. If no class NC request is in the system when the last class C stage departs, K additional stages of class C work will be scheduled.
(2) exponential event for the arrival of a class NC request:

$$(i, j, w) \xrightarrow{\lambda} (i, j+1, w), \qquad (j < B) \tag{23}$$

(3) exponential event for the departure of a class NC request:

$$(0, j, w) \xrightarrow{\beta} (0, j-1, w), \qquad (j > 1) \tag{24}$$
$$(0, 1, w) \xrightarrow{\beta} (K, 0, w+1), \qquad (w < N_{mc}) \tag{25}$$
$$(0, 1, N_{mc}) \xrightarrow{\beta} (0, 0, N_{mc}), \tag{26}$$

Again, Equation (25) indicates the greedy characteristic of the algorithm. If there are no class NC requests in the system, K additional stages of class C work are scheduled.
(4a) for the *greedy* strategy, the deterministic event corresponding to the end of the k^{th} mini-cycle in each period of length T (where $k \in \{1, 2, \ldots, N_{mc}-1\}$):

$$(i,j,w) \longrightarrow (i+K, j, w+1), \qquad (w < N_{mc}) \tag{27}$$
$$(i,j,N_{mc}) \longrightarrow \quad (i,j,N_{mc}), \tag{28}$$

(4b) for the *partially greedy* strategy, the deterministic event corresponding to the end of the k^{th} mini-cycle in each period of length T (where $k \in \{1,2,\ldots,N_{mc}-1\}$):

$$(i,j,w) \longrightarrow (i+K, j, w+1), \qquad (w \leq k) \tag{29}$$
$$(i,j,w) \longrightarrow \quad (i,j,w), \qquad (k < w \leq N_{mc}) \tag{30}$$

Equations (27) to (30) illustrate the difference between the greedy and the partially greedy strategies.
(5) for the *greedy* and the *partially greedy* strategy, the deterministic event corresponding to the end of the last mini-cycle in a period of length T:

$$(i,j,N_{mc}) \longrightarrow (K,j,1), \tag{31}$$

Note that in Equation (31), the states (i,j,N_{mc}) where $i > 0$ correspond to the occurrence of the overflow condition at the end of a period of length T.

3.2.2 Model for the Greedy Gated Algorithm

Similar to the non-greedy algorithm, when class NC requests are served under the gated discipline, two additional variables are added to the state space. Specifically, in addition to the total number of class NC requests, j, we include h, the number of class NC requests which are still in the system *and* were present when the gate closed, and g, the number of class NC requests present at the last gate closure (which is used to determine the service rate β_g of class NC requests). Thus the states of the gated model are of the form (i,j,w,h,g). As in the non-greedy case, we have $h \leq j$ and $h \leq g$, while we also have $w \geq k$ where k is the number of the current mini-cycle. Therefore, $i \in \{0,\ldots,N_{mc}K\}$, $j,h,g \in \{0,\ldots,B\}$, and $w \in \{1,\ldots,N_{mc}\}$. The events of interest include:
(1) exponential event for the departure of a class C stage:

$$(i,j,w,0,0) \xrightarrow{\alpha} (i-1,j,w,0,0), \qquad (i>1) \tag{32}$$
$$(1,j,w,0,0) \xrightarrow{\alpha} \quad (0,j,w,j,j), \qquad (j>0) \tag{33}$$
$$(1,0,w,0,0) \xrightarrow{\alpha} (K,0,w+1,0,0), \qquad (w < N_{mc}) \tag{34}$$
$$(1,0,N_{mc},0,0) \xrightarrow{\alpha} \quad (0,0,N_{mc},0,0), \tag{35}$$

Equation (33) indicates the gated policy and corresponds to the gate closure instant. Equation (34) indicates the greedy characteristic for which K additional stages of class C work are introduced into the system.
(2) exponential event for the arrival of a class NC request:

$$(i,j,w,h,g) \xrightarrow{\lambda} (i,j+1,w,h,g), \qquad (j < B) \tag{36}$$

(3) exponential event for the departure of a class NC request:

$$(0,j,w,h,g) \xrightarrow{\beta_g} (0,j-1,w,h-1,g), \qquad (h > 1) \tag{37}$$
$$(0,j,w,1,g) \xrightarrow{\beta_g} (0,j-1,w,j-1,j-1), \qquad (j > 1) \tag{38}$$
$$(0,1,w,1,g) \xrightarrow{\beta_g} (K,0,w+1,0,0), \qquad (w < N_{mc}) \tag{39}$$
$$(0,1,N_{mc},1,g) \xrightarrow{\beta_g} (0,0,N_{mc},0,0), \tag{40}$$

Equation (38) corresponds to the gate closing so as to serve the next batch of class NC requests. Equation (39) illustrates the greedy characteristic of the algorithm.

(4a) for the *greedy* strategy, the deterministic event for the end of the k^{th} mini-cycle in each period of length T where $k \in \{1,2,\ldots,N_{mc}-1\}$:

$$(i,j,w,0,0) \longrightarrow (i+K,j,w+1,0,0), \qquad (i > 0) \wedge (w < N_{mc}) \tag{41}$$
$$(i,j,N_{mc},0,0) \longrightarrow (i,j,N_{mc},0,0), \qquad (i > 0) \tag{42}$$
$$(0,j,w,h,g) \longrightarrow (K,j,w+1,0,0), \qquad (w < N_{mc}) \tag{43}$$
$$(0,j,N_{mc},h,g) \longrightarrow (0,j,N_{mc},h,g), \tag{44}$$

(4b) for the *partially greedy* strategy, the deterministic event for the end of the k^{th} mini-cycle in each period of length T where $k \in \{1,2,\ldots,N_{mc}-1\}$:

$$(i,j,w,0,0) \longrightarrow (i+K,j,w+1,0,0), \qquad (i > 0) \wedge (w \leq k) \tag{45}$$
$$(i,j,w,0,0) \longrightarrow (i,j,w,0,0), \qquad (i > 0) \wedge (k < w \leq N_{mc}) \tag{46}$$
$$(0,j,w,h,g) \longrightarrow (K,j,w+1,0,0), \qquad (w \leq k) \tag{47}$$
$$(0,j,w,h,g) \longrightarrow (0,j,w,h,g), \qquad (k < w \leq N_{mc}) \tag{48}$$

Equations (41) to (48) illustrate the difference between the greedy and partially greedy strategies.

(5) for the *greedy* and the *partially greedy* strategy, the deterministic event for the end of the last mini-cycle in a period of length T:

$$(0,j,N_{mc},h,g) \longrightarrow (K,j,1,0,0), \tag{49}$$
$$(i,j,N_{mc},0,0) \longrightarrow (K,j,1,0,0), \qquad (i > 0) \tag{50}$$

Since no overrun is allowed, Equations (49) and (50) indicate that the system will restart with K stages of class C work. It is important to point out that states indicated by Equation (50) are the states for which a class C overflow has occurred.

In this section we consider the calculation of two main performance measures: (1) the probability of overflow for class C requests, and (2) the number of class NC requests in the system averaged over all time. The first of these measures is simply the probability that at least one stage of the class C workload which is introduced into the system during a period of length T is still present in the system at the end of this time period. For models without overruns, which are studied in this section, such stages are dropped from the system. Models with overruns are considered in [6]. A related measure is the probability that a class C request will meet its deadline. The second performance measure, the time average number of class NC requests in the system, can also be used to find other measures of interest. Specifically, the mean response time of a class NC request can be obtained from it with an application of Little's Result.

3.3.1 Probability of Overflow

We first consider the probability of overflow for the NG-FCFS and the NG-Gated algorithms. In this case, class NC requests do not affect the behavior of class C stages, so that the overflow probability under both service disciplines is identical. Assuming that overruns are not allowed, the number of stages at the beginning of any period $[\ell T, (\ell+1)T]$ is exactly K. We wish to determine the probability that at the end of the period of length T at least one class C stage has not been completely served and remains in the system. The period is split into N_{mc} mini-cycles, each of length T/N_{mc}. We will find the distribution of the number of class C stages at the beginning of each mini-cycle and at the end of each mini-cycle. Let $\xi^{(k)}$, $k = 1, \ldots, N_{mc}$, be the state probability vector of the number of class C stages in the system at the *start* of the k^{th} mini-cycle. Similarly, let $\chi^{(k)}$, $k = 1, \ldots, N_{mc}$, be the state probability vector of the number of class C stages in the system at the *end* of the k^{th} mini-cycle. The size of $\xi^{(k)}$ is $1 \times (kK+1)$, and the size of $\chi^{(k)}$ is also $1 \times (kK+1)$, as will be shown below.

Clearly $\xi^{(1)} = \mathbf{e}_K$, where the K^{th} entry of $\mathbf{e}_K$ is 1 and all other entries are 0. This holds since K stages are always present at the start of the first mini-cycle. To find $\chi^{(1)}$, the probabilities at the end of the first mini-cycle, we consider a pure death process with states $0, 1, \ldots, K$, which represent the number of stages in the system during the first mini-cycle (see Figure 4 with $k = 1$). The

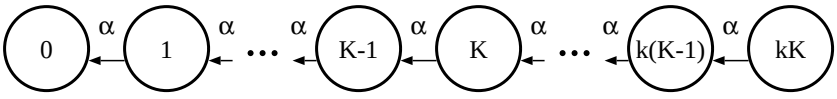

Fig. 4. Pure Death Chain for Overflow Probability

rate from state i to $i-1$ ($i = 1, \ldots, K$) is α, the service rate for a class C stage. State 0 is an absorbing state, since once there are no stages in the system, no additional stages of class C workload can arrive (or be served) during the mini-cycle for the non-greedy algorithms. To find the probabilities of the number of stages at the end of the mini-cycle, we use uniformization [12,13] on the pure death chain over a period of length T/N_{mc}. Alternatively, note that a transition from K at the beginning of the mini-cycle to $i' > 0$ at the end of the mini-cycle occurs if and only if there are $K - i'$ departures during time T/N_{mc} in the pure death chain, i.e., with probability $e^{-\alpha T/N_{mc}}(\alpha T/N_{mc})^{K-i'}/(K-i')!$. The probability of a transition from K to 0 is the probability of at least K departures, since 0 is an absorbing state. Thus we obtain the $1 \times (K+1)$ vector $\chi^{(1)}$ for the state probabilities at the end of the first mini-cycle.

To find the number of stages at the beginning of the second mini-cycle, we simply add K stages to those (if any) that are present at the end of the first mini-cycle. Thus $\xi^{(2)}$ is the $1 \times (2K+1)$ vector obtained by taking $\chi^{(1)}$ and shifting it K entries to the right. That is,

$$\xi_i^{(2)} = 0, \qquad i = 0, \ldots, K-1 \tag{51}$$

$$\xi_i^{(2)} = \chi_{i-K}^{(1)}, \qquad i = K, \ldots, 2K. \tag{52}$$

To find the state probability vector $\chi^{(2)}$ at the end of the second mini-cycle, we consider a pure death chain with states $0, \ldots, 2K$, with state 0 being an absorbing state (see Figure 4 with $k = 2$). The initial number i of stages can be any of $K, \ldots, 2K$, and its distribution is given by $\xi^{(2)}$, while the final number i' at the end of the second mini-cycle can be any of $0, \ldots, 2K$ with distribution $\chi^{(2)}$. We use uniformization on this pure death chain with $2K+1$ states for a time period of T/N_{mc}. Alternatively, a transition from i to $i' > 0$ occurs if there are $i - i'$ departures during T/N_{mc}, while a transition from i to 0 occurs if there are at least i departures. This gives $\chi^{(2)}$.

The $1 \times (3K+1)$ vector $\xi^{(3)}$ corresponding to the start of the third mini-cycle is obtained by shifting $\chi^{(2)}$ to the right K entries and filling the first K entries with zeros, as before. Continuing in this manner, each time considering a pure death chain with K more states, we finally obtain $\chi^{(N_{mc})}$, the state probability vector at the end of mini-cycle N_{mc}. That is, $\chi^{(N_{mc})}$ is the state probability vector for the end of the period of length T. Note that $\chi^{(N_{mc})}$ is $1 \times (N_{mc}K+1)$, since any number $0, \ldots, N_{mc}K$ of class C stages may occur at the end of the period of length T. The probability of overflow is then simply

$$P[\text{overflow}] = \sum_{i=1}^{N_{mc}K} \chi_i^{(N_{mc})} = 1 - \chi_0^{(N_{mc})} \tag{53}$$

This is the probability that at least one class C stage remains in the system

at the end of the period of length T. Note that a closed-form expression for the probability of overflow is available in terms of Poisson probabilities.

3.3.2 Time Averages for Class NC

We now wish to find the distribution of the number of class NC requests in the system averaged over all time. In obtaining this measure, the class C requests do affect class NC requests, so we will have to keep track of both classes. In the NG-Gated case, one must keep both original class NC requests and total class NC requests along with an indicator of the rate β_g at which class NC requests depart the system.

Let $X(t)$, $t \geq 0$, be the stochastic process with state space $\mathcal{S}$ which represents the behavior of the NG-FCFS algorithm without overruns. We outline the procedure for finding time average distributions corresponding to $X(t)$. Let $\mathcal{L} \subset \mathcal{S}$ be a subset of states, and suppose we wish to find $P_{\mathcal{L}} = \lim_{t\to\infty} P[X(t) \in \mathcal{L}]$. A particular case of interest occurs when the subset consists of a single state s (i.e., $\mathcal{L} = \{s\}$), and we use the simplified notation $P_s = \lim_{t\to\infty} P[X(t) = s]$.

Consider the time points that are the beginnings of mini-cycles. Then using Markov chains with rewards, it can be shown as in [9] that

$$P_{\mathcal{L}} = \frac{1}{T/N_{mc}} \sum_{s\in\mathcal{S}} E[L_s]\delta_s. \tag{54}$$

Here $\boldsymbol{\delta}$ is the stationary distribution for $X(t)$ at the start of a mini-cycle, with δ_s being the entry corresponding to state $s \in \mathcal{S}$. The random variable L_s represents the amount of time during a mini-cycle that $X(t)$ spends in states of $\mathcal{L}$ given that the initial state at the start of the mini-cycle was s. Note that a variable k, $k = 1, \ldots, N_{mc}$, must be included in the state description to refer to the appropriate mini-cycle. Letting $\mathbf{H}$ be the matrix that gives the transition probabilities from the start of a mini-cycle to the start of the next mini-cycle, then it follows that $\mathbf{H}$ has a cyclic structure. Specifically, we have

$$\mathbf{H} = \begin{bmatrix} 0 & \mathbf{H}_1\mathbf{G}_1 & 0 & \cdots & 0 \\ 0 & 0 & \mathbf{H}_2\mathbf{G}_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \mathbf{H}_{N_{mc}-1}\mathbf{G}_{N_{mc}-1} \\ \mathbf{H}_{N_{mc}}\mathbf{G}_{N_{mc}} & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{55}$$

Here $\mathbf{H}_k$ is the matrix that gives the transition probabilities from the start of

the k^{th} mini-cycle to its end, while $\mathbf{G}_k$ gives the transition probabilities from the end of the k^{th} mini-cycle to the start of the $(k+1)^{st}$ mini-cycle (mod N_{mc}). Thus $\mathbf{G}_k$ simply accounts for the possible introduction of K class C stages at a mini-cycle boundary.

An expression for $P_{\mathcal{L}}$ can also be found by considering time points corresponding to the beginning of the first mini-cycle in a period of length T. More generally, one can consider the $2N_{mc}$ different sequences of points that represent (a) the beginning of mini-cycle k or (b) the end of mini-cycle k, $k = 1, \ldots, N_{mc}$. Let $\boldsymbol{\eta}^{(k)}$, $k = 1, \ldots, N_{mc}$, be the stationary probability vector for $X(t)$ at the beginning of the k^{th} mini-cycle, and similarly let $\boldsymbol{\nu}^{(k)}$, $k = 1, \ldots, N_{mc}$, correspond to the end of the k^{th} mini-cycle. Now note that

$$\boldsymbol{\eta}^{(1)} = \boldsymbol{\eta}^{(1)}\mathbf{H}_1\mathbf{G}_1 \cdots \mathbf{H}_{N_{mc}}\mathbf{G}_{N_{mc}}$$

and

$$\boldsymbol{\nu}^{(1)} = \boldsymbol{\nu}^{(1)}\mathbf{G}_1\mathbf{H}_2 \cdots \mathbf{G}_{N_{mc}}\mathbf{H}_1.$$

Multiplying the first equation on the right by $\mathbf{H}_1$, we see that $\boldsymbol{\eta}^{(1)}\mathbf{H}_1$ also satisfies the second equation. Thus, assuming irreducibility, we have equality of these probability vectors, $\boldsymbol{\nu}^{(1)} = \boldsymbol{\eta}^{(1)}\mathbf{H}_1$. Similarly, $\boldsymbol{\eta}^{(2)} = \boldsymbol{\nu}^{(1)}\mathbf{G}_1$, and in general $\boldsymbol{\eta}^{(k+1)} = \boldsymbol{\nu}^{(k)}\mathbf{G}_k \ (\mathrm{mod}\, N_{mc})$, $\boldsymbol{\nu}^{(k)} = \boldsymbol{\eta}^{(k)}\mathbf{H}_k$. When considering quantities over periods of length T, the variable k does not need to appear as part of the state, since this information is implicit in the above numbering scheme. We then have

$$P_{\mathcal{L}} = \frac{1}{T}\sum_{s\in\mathcal{S}} E[U_s^{(k)}]\eta_s^{(k)}, \qquad k = 1, \ldots, N_{mc} \tag{56}$$

where $U_s^{(k)}$ is the amount of time spent in $\mathcal{L}$ from the beginning of the k^{th} mini-cycle to the beginning of the next such k^{th} mini-cycle, given the initial state was s. Similarly,

$$P_{\mathcal{L}} = \frac{1}{T}\sum_{s\in\mathcal{S}} E[V_s^{(k)}]\nu_s^{(k)}, \qquad k = 1, \ldots, N_{mc} \tag{57}$$

where $V_s^{(k)}$ is defined in terms of the end of the k^{th} mini-cycle.

Suppose we wish to find the time average distribution for a subset $\mathcal{L}$ by focusing on the start of the first mini-cycle of a period of length T, i.e., we set $k = 1$ in Equation (56). Then the expected time spent in $\mathcal{L}$ during the period T is

the sum of the corresponding expectations over the N_{mc} mini-cycles. That is,

$$E[U_s^{(1)}] = \sum_{\ell=1}^{N_{mc}} E[U_s^{(1)}(\ell)], \tag{58}$$

where $U_s^{(1)}(\ell)$ is the amount of time spent in $\mathcal{L}$ during the ℓ^{th} mini-cycle given the initial state s at the start of the period T. Thus

$$P_{\mathcal{L}} = \frac{1}{T} \sum_{s \in \mathcal{S}} \sum_{\ell=1}^{N_{mc}} E[U_s^{(1)}(\ell)] \eta_s^{(1)}. \tag{59}$$

Now $E[U_s^{(1)}(\ell)]$ is found by using uniformization on an interval of length T/N_{mc}. Recall that $X(t)$ is Markovian during the ℓ^{th} mini-cycle with a finite state space, so its behavior during that time can be obtained using uniformization. We let $\mathbf{W}^{(\ell)}$ be the transition probability matrix of the resulting discrete-time chain, and Λ_ℓ be the uniformization rate. Given n transitions of the uniformized chain, the mini-cycle is split into $n+1$ subintervals. It is well-known that the subinterval lengths are exchangeable random variables [5], and so each subinterval has expected length $(T/N_{mc})/(n+1)$. To find the expected amount of time the uniformized process $X(t)$ spends in the subset $\mathcal{L}$ in the ℓ^{th} mini-cycle, we need only multiply the probability that the discrete-time chain is in $\mathcal{L}$ at the m^{th} step $(m = 0, \ldots, n)$ by the expected subinterval length and sum over m. Unconditioning on the number of transitions, we have

$$E[U_s^{(1)}(\ell)] = \sum_{n=0}^{\infty} e^{-\Lambda_\ell T/N_{mc}} \frac{(\Lambda_\ell T/N_{mc})^n}{n!} \left(\frac{T/N_{mc}}{n+1} \right) \sum_{m=0}^{n} \sum_{s' \in \mathcal{L}} \pi_{s'}^{\ell}(m, \zeta_s) \tag{60}$$

where $\boldsymbol{\pi}^\ell(m, \zeta_s) = \boldsymbol{\pi}^\ell(m-1, \zeta_s)\mathbf{W}^{(\ell)}$ gives the probability vector at step m for the discrete-time chain obtained by uniformizing $X(t)$ over the ℓ^{th} mini-cycle, and $\boldsymbol{\pi}^\ell(0, \zeta_s) = \zeta_s$ is the s^{th} row of the matrix $\mathbf{H}_1\mathbf{G}_1 \cdots \mathbf{H}_\ell\mathbf{G}_\ell$ that gives the transitions from the start of the first mini-cycle to the start of mini-cycle ℓ.

Now note that

$$\sum_{s \in \mathcal{S}} \boldsymbol{\pi}^\ell(m, \zeta_s)\eta_s^{(1)} = \sum_{s \in \mathcal{S}} \boldsymbol{\pi}^\ell(0, \zeta_s)[\mathbf{W}^{(\ell)}]^m \eta_s^{(1)} = \boldsymbol{\eta}^{(1)}\mathbf{H}_1\mathbf{G}_1 \cdots \mathbf{H}_\ell\mathbf{G}_\ell[\mathbf{W}^{(\ell)}]^m,$$

and since $\boldsymbol{\eta}^{(1)}\mathbf{H}_1\mathbf{G}_1 \cdots \mathbf{H}_\ell\mathbf{G}_\ell = \boldsymbol{\eta}^{(\ell)}$ we have

$$\sum_{s \in \mathcal{S}} \boldsymbol{\pi}^\ell(m, s)\eta_s^{(1)} = \boldsymbol{\eta}^{(\ell)}[\mathbf{W}^{(\ell)}]^m = \boldsymbol{\pi}^\ell(m, \boldsymbol{\eta}^{(\ell)}).$$

It follows from (59) and (60) that

$$P_{\mathcal{L}} = \frac{1}{T} \sum_{\ell=1}^{N_{mc}} \sum_{n=0}^{\infty} e^{-\Lambda_\ell T/N_{mc}} \frac{(\Lambda_\ell T/N_{mc})^n}{n!} \left(\frac{T/N_{mc}}{n+1} \right) \sum_{m=0}^{n} \sum_{s' \in \mathcal{L}} \pi_{s'}^{\ell}(m, \boldsymbol{\eta}^{(\ell)}).$$

Thus the initial vector corresponding to the uniformization over the ℓ^{th} mini-cycle is the stationary distribution $\boldsymbol{\eta}^{(\ell)}$ for the beginning of that mini-cycle. When the *same* uniformized chain is used in all mini-cycles, we may write $\Lambda_\ell = \Lambda$, $\mathbf{W}^{(\ell)} = \mathbf{W}$ and $\boldsymbol{\pi}^{\ell} = \boldsymbol{\pi}$, so that the above results simplify to

$$P_{\mathcal{L}} = \sum_{n=0}^{\infty} e^{-\Lambda T/N_{mc}} \frac{(\Lambda T/N_{mc})^n}{n!} \left\{ \frac{\sum_{m=0}^{n} \sum_{s' \in \mathcal{L}} \sum_{\ell=1}^{N_{mc}} \pi_{s'}(m, \boldsymbol{\eta}^{(\ell)})}{(n+1)N_{mc}} \right\}$$

Finally, since

$$\frac{1}{N_{mc}} \sum_{\ell=1}^{N_{mc}} \boldsymbol{\pi}(m, \boldsymbol{\eta}^{(\ell)}) = \frac{1}{N_{mc}} \sum_{\ell=1}^{N_{mc}} \boldsymbol{\pi}(0, \boldsymbol{\eta}^{(\ell)}) \mathbf{W}^m = \frac{\boldsymbol{\eta}^{(1)} + \cdots + \boldsymbol{\eta}^{(N_{mc})}}{N_{mc}} \mathbf{W}^m,$$

we may perform a single uniformization with the probability vector

$$\hat{\boldsymbol{\eta}} \triangleq \frac{1}{N_{mc}} (\boldsymbol{\eta}^{(1)} + \cdots + \boldsymbol{\eta}^{(N_{mc})}),$$

instead of carrying out N_{mc} instances of uniformization. The final expression is

$$P_{\mathcal{L}} = \sum_{n=0}^{\infty} e^{-\Lambda T/N_{mc}} \frac{(\Lambda T/N_{mc})^n}{n!} \left\{ \frac{\sum_{m=0}^{n} \sum_{s' \in \mathcal{L}} \pi_{s'}(m, \hat{\boldsymbol{\eta}})}{n+1} \right\}. \tag{61}$$

The vectors $\boldsymbol{\eta}^{(\ell)}$ and $\boldsymbol{\nu}^{(\ell)}$, $\ell = 1, \ldots, N_{mc}$, can be obtained simultaneously by iterating from mini-cycle to mini-cycle. This is equivalent to employing $2N_{mc}$ simultaneous occurrences of the power method [22]. Note that, up to a normalization constant, the vector $[\boldsymbol{\eta}^{(1)}, \ldots, \boldsymbol{\eta}^{(N_{mc})}]$ is the stationary distribution for the cyclic matrix $\mathbf{H}$. Thus we are, in fact, finding the stationary distribution for the Markov chain corresponding to the embedded points that are the beginning of a (any) mini-cycle.

We now return to the consideration of the mixed workload model. Suppose we wish to find the time average distribution for the number of class NC requests in the system. In the NG-FCFS case with no overruns, the states of $X(t)$ are of the form $s = (i, j)$. For $j' \in \{0, \ldots, B\}$, we consider the subset of states $\mathcal{L}_{j'} = \{(i, j') : i = 0, \ldots, N_{mc}K\}$. Applying the above results to $\mathcal{L}_{j'}$ will yield the limiting probability that there are j' requests of class NC in

the system. In the case of no overruns, there are always K class C stages at the beginning of the period of length T. Unlike the calculation of the overflow probability, class C influences the class NC requests, and so (for NG-FCFS) we also have to keep track of class C stages. The process $X(t)$ is Markovian during each mini-cycle, and the solution proceeds using uniformization, except that the pure death Markov chain of the overflow probability case is replaced by a two-dimensional chain that keeps track of both classes. The chain for the first mini-cycle has $(B+1)(K+1)$ reachable states, since only at most K class C stages can be present during this mini-cycle. Each state has the form (i,j), $i=0,\ldots,K$, $j=0,\ldots,B$, where i represents the class C stages and j represents the class NC requests. Transitions out of a state (i,j) can occur to state $(i-1,j)$ at rate α $(i>0)$, which corresponds to the completion of service of a class C stage; or to state $(i,j+1)$ at rate λ $(j<B)$, which corresponds to the arrival of a class NC request; or a transition can occur from $(0,j)$ with $j>0$ at rate β to $(0,j-1)$, which corresponds to the completion of service of a class NC request. Figure 5 illustrates this chain for the case of mini-cycle k, $k=1,\ldots,N_{mc}$.

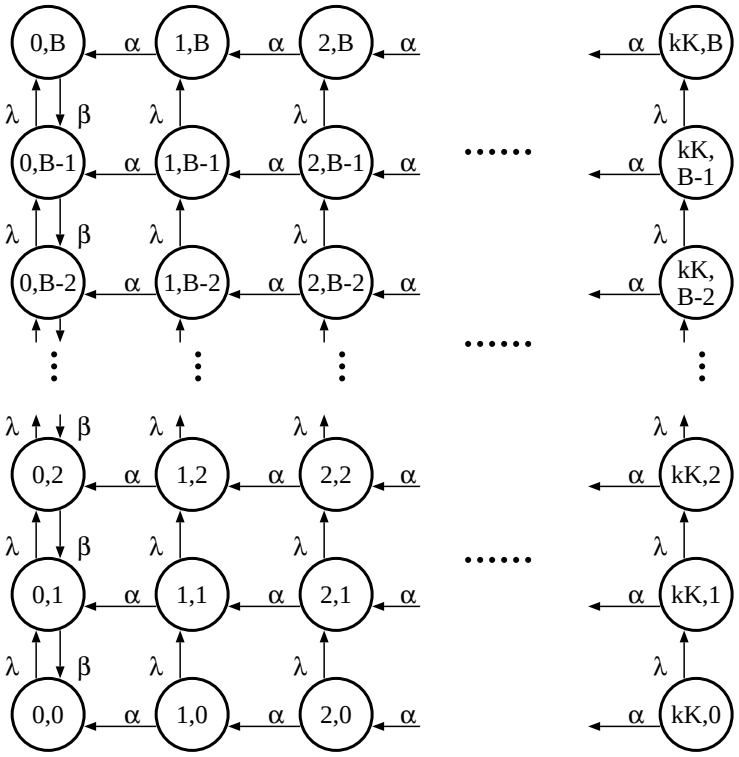

Fig. 5. Markov Chain for Embedded Points

Suppose an initial distribution corresponding to the start of the first mini-cycle, say $\boldsymbol{\eta}^{(1)}(0)$, is given. After using uniformization on the chain in Figure 5 (with $k=1$) for time T/N_{mc}, one obtains the state probability vector $\boldsymbol{\nu}^{(1)}(0)$. To find $\boldsymbol{\eta}^{(2)}(0)$, which is $1\times(B+1)(2K+1)$, one takes the $1\times(B+1)(K+1)$ vector $\boldsymbol{\nu}^{(1)}(0)$ and shifts it right by K columns. That is,

$$\boldsymbol{\eta}^{(2)}_{i,j}(0) = 0, \qquad i = 0, \ldots, K-1, j = 0, \ldots, B$$
$$\boldsymbol{\eta}^{(2)}_{i,j}(0) = \boldsymbol{\nu}^{(1)}_{i-K,j}(0), \qquad i = K, \ldots, 2K, j = 0, \ldots, B$$

We now use this as the initial distribution for the Markov chain with states (i, j), $i = 0, \ldots, 2K$, $j = 0, \ldots, B$, and apply uniformization to obtain $\boldsymbol{\nu}^{(2)}(0)$. Continuing in this way, we finally obtain $\boldsymbol{\nu}^{(N_{mc})}(0)$, the state probability vector at the end of mini-cycle N_{mc} given the starting vector $\boldsymbol{\eta}^{(1)}(0)$. Iterating we obtain the stationary vectors $\boldsymbol{\eta}^{(k)}$ and $\boldsymbol{\nu}^{(k)}$. Then we can apply the previous time average results to the subset $\mathcal{L}_{j'}$ to find the distribution of time for which there are j' class NC requests in the system. From this one can obtain the mean number and, via Little's Result, the mean response time. Also, the throughput may be calculated by determining the rate of arrivals of class NC requests to the system.

The analysis of the partially greedy and greedy algorithms is similar, except that the Markov chains between the start and end of a mini-cycle are more complicated than in the non-greedy case. For example, to find the overflow probability, the class NC requests must be taken into account, since class C stages may be introduced into the system if no class NC requests are being served. Thus the one-dimensional chain of the non-greedy case is replaced with a larger chain. Further, the gated case (even for non-greedy) is more complicated than FCFS, since the total number of NC requests, the number of original requests left who can be served, and an indicator g of the service rate β_g must all be kept. However, the method of analysis is again based on embedded Markov chains and uniformization, although the individual chains may become larger as the algorithm becomes more complicated.

4 Numerical Results

In this section, we illustrate the use of the methodology of Section 3 for evaluating mixed workload scheduling algorithms through numerical examples. We begin with the details of workload model and parameter estimation for class C and class NC requests, using the Erlang and the exponential distributions, respectively. Specifically, in the examples of this section we consider a system which employs disks whose characteristics are given in Table 1. Note that the seek time is in seconds, and it is a function of the request seek distance d.

4.1 Workload Characterization

We set a requirement that each disk in the system has to support $N_c = 24$ class C requests, which represent MPEG-1 streams with an average display

Disk capacity	2.25 GBytes
Number of cylinders	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

rate of 1.5 Mbps each. The average transfer size of each NC request is 46.875 KBytes. We assume that the QoS requirement for the class C requests is that the probability of an overloaded cycle has to be less than or equal to 0.01 (i.e., Prob$[\tau_{N_c} \geq T] \leq 0.01$). Given the disk characteristics in Table 1 and this QoS requirement, we can solve Equation (4) numerically to determine the minimum value of T such that Prob$[\tau_{N_c} \geq T] \leq 0.01$, i.e., $T = 1.48754$ seconds.

We model the transfer size of a class C request as an exponential distribution with a mean of 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 MPEG-1 stream). Therefore, the mean transfer time $t_{tfr} = E[\tau_{tfr}]$ is equal to 0.03 secs. The rotational latency for both class C and class NC requests is modeled as a uniform distribution in the range of $[0, t_r] = [0, 8.33]$ milliseconds, where t_r is the maximum latency. As already stated, the seek time is assumed to be deterministic (see Section 2) and can be obtained using Table 1. Thus, for class C requests, Equation (3) gives the Laplace transform of the total service time requirement of the N_c continuous requests, i.e.,

$$F^*_{N_c}(s) = e^{-s\ \tau^{max}_{seek}(N_c)} \left[\left(\frac{1 - st_r}{st_r} \right) \left(\frac{1}{1 + st_{tfr}} \right) \right]^{N_c}. \tag{62}$$

The mean, $E[\tau_{N_c}]$, and the variance, σ_{N_c}, of the total service time of N_c class C requests are

$$E[\tau_{N_c}] = \tau^{max}_{seek}(N_c) + N_c \left(t_r/2 + t_{tfr} \right) \tag{63}$$

$$\sigma_{\tau_{N_c}} = N_c \left(t_r^2/12 + t_{tfr}^2 \right). \tag{64}$$

To determine the parameters for the class C workload, we observe that the coefficient of variation is always less than 1.0 for $N_c \geq 1$. Thus, we use an Erlang distribution to represent the class C workload. Given the value of N_c, we compute the appropriate number of stages in an Erlang distribution so as to match the mean in Equation (63) and match the variance in Equation (64) to within a 5% deviation.

What remains is to determine the parameters of the NC workload. Under the

FCFS service discipline of class NC requests (assuming that the data blocks are uniformly distributed on a disk surface), we use an average seek distance of 1/3 of the maximum seek distance. Thus, using the seek time function in Table 1, we obtain the *average* seek time of 0.009275 secs. The rotational latency is uniformly distributed with an average of 0.00425 secs. Since the average transfer size of an NC request is 46.875 KBytes, the average transfer latency is 0.004882 secs. Thus, we model the service time of an NC request as an *exponential* distribution with a mean of 0.018407 secs (which includes seek, rotation, and transfer time).

The only difference in obtaining the parameters corresponding to gated service of class NC requests is in determining the average seek time, which in general, will be smaller than in the FCFS case. Specifically, the average seek distance for *a gated group* of class NC requests with a group size of $g \geq 1$ is given by

$$d_{seek}^{g} = \left\lceil d_{seek}^{full} * \left[\left(\frac{2}{g+1} \right) \left(\frac{g}{g+2} \right) + \left(\frac{g-1}{g+1} \right) \left(\frac{g}{g+2} \right) \left(\frac{3}{2} \right) \right] \right\rceil , \qquad (65)$$

where d_{seek}^{full} denotes the longest possible seek distance (in number of cylinders) of a disk. The above expression can be obtained as follows. First, observe that after finishing service of the continuous requests, the disk head is positioned in a random place on the disk. Thus, in general, the disk may have to serve the g class NC requests by doing sweeps in both directions. That is, the g requests are served by sorting them based on the current *location* (i.e., where the disk head stopped after the last group of served requests) and *direction* (i.e., the direction in which the previous sweep was going) of the disk head and then retrieved in that order. Secondly, we make another simplifying assumption, namely, that the g requests together with the disk head are uniformly distributed on the surface of the disk, i.e., they correspond to $(g+1)$ equally spaced points that divide the disk surface into $(g+2)$ partitions of equal size. Then the first term in Equation (65) represents all the cases for which 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 $\left(\frac{g}{g+2} d_{seek}^{full}\right)$ to serve these g class NC requests, and these cases occur with probability $\left(\frac{2}{g+1}\right)$. The second term in Equation (65) represents all the cases for which the disk is not on an edge of the disk surface. In such cases, the disk head needs to seek a distance of $(\frac{3}{2})(\frac{g}{g+2}) d_{seek}^{full}$, and these cases occur with probability $\left(\frac{g-1}{g+1}\right)$. Once the average seek distance is found, the average seek time of each class NC request can be obtained using Table 1, and the total service time of each class NC request is again modeled as an *exponential* distribution.

We now present several numerical results, which are all computed through the use of the methodology described in Section 3, to illustrate the type of performance tradeoffs and design decisions that can be studied using our analytical models. Specifically, in these examples, for the greedy variations on the algorithms, we use the *partially greedy* algorithms. In all cases we allow overruns, using the overrun extension to the models of Section 3 (for details refer to [6]). Thus, the results given below are for fairly complex variations of the models. Lastly, we present performance results for both classes of customers. All numerical results were obtained by implementing the methodology described in this paper using the Tangram-II modeling tool [3].

Figure 6 depicts the probability of overflow of class C customers as a function of the arrival rate of class NC customers, where $N_{mc} = 6$. As noted in Section

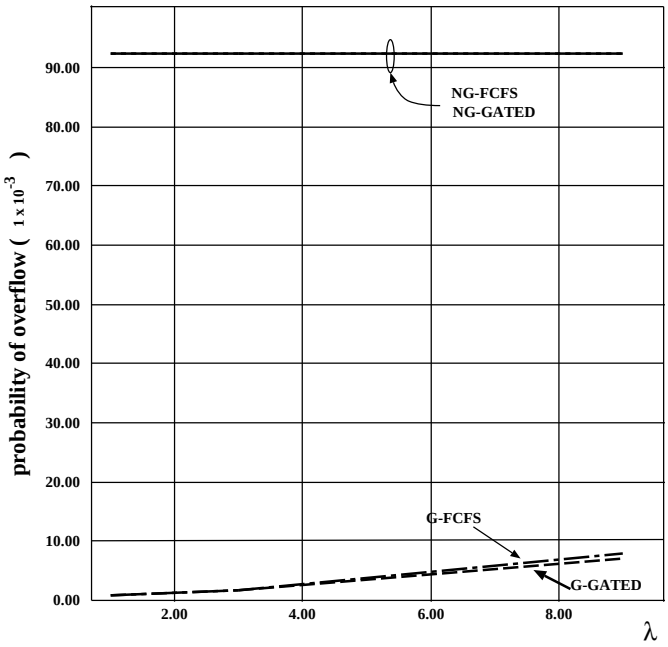

Fig. 6. Probability of overflow of class C requests as a function of arrival rate of class NC requests.

3.3, the probability of overflow in the *non-greedy* algorithms is independent of the arrival rate as well as the service discipline of class NC customers; this, of course, is not the case for the *greedy* algorithms. Note also, that the greedy algorithms are able to sustain the QoS requirement (i.e., that of $p = 0.01$) at $N_{mc} = 6$, whereas the non-greedy algorithms are not able to provide the required QoS at such a high number of mini-cycles in this experiment. Recall that the value of T is computed based on the assumption of $N_{mc} = 1$. As we already noted, it is expected that with higher values of N_{mc} the probability of overflow will increase (as there are fewer opportunities for seek optimization

and hence the overall service demand of N_c requests is increased). We further elaborate on this point below.

Figure 7 illustrates the expected response time for class NC customers as a function of their arrival rate. The results in this figure for the greedy algorithms

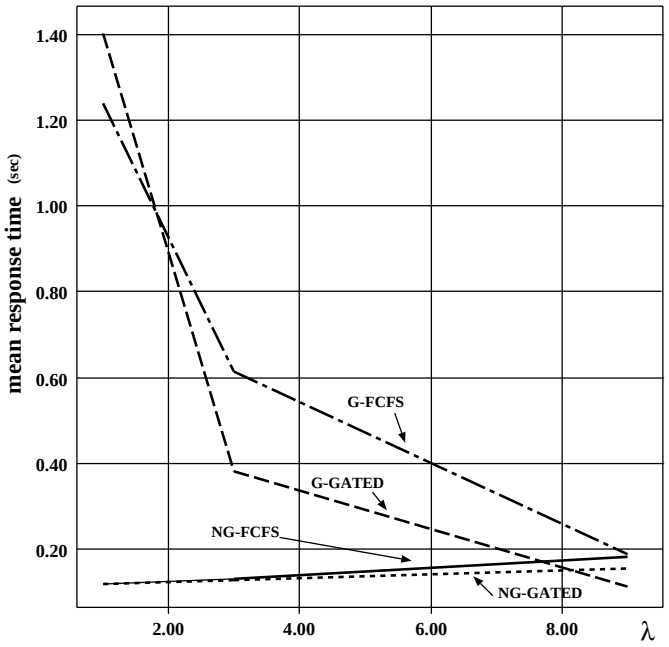

Fig. 7. Expected response time of NC requests as a function of the NC requests arrival rate.

are fairly unusual and deserve a few words of explanation. Recall that the buffer size for class NC requests is finite, and this finite buffer size (when reasonably small) produces the unusual effect. Note that class NC requests that arrive during service of class C requests tend to experience longer response times; however, those that arrive during service of other class NC requests can experience a fairly short response time (assuming a mini-cycle does not expire before they receive service). These short response times, have a higher probability of occurring at higher arrival rates. At the lower arrival rates there is little chance that more customers will arrive to take advantage of the server, and thus in the greedy case the server will resume service of class C requests (contributing to longer response times of class NC requests). Note also that we are computing the expected response time for the class NC customers that are "accepted" into the system (i.e., those that arrive when the buffer is not full). Hence, at the higher arrival rates, the queue does not build up sufficiently (due its finite capacity) to contribute significantly to increases in mean response time. The degree of significance of all these effects depends on the actual buffer size, and this unusual behavior will disappear as the maximum buffer size for the class NC requests is increased. However, if it is desirable to maintain small buffer sizes in the real system, then this effect

needs to be considered and studied further in order to make appropriate design choices.

Lastly, Figures 8 and 9 depict the probability of overflow of class C requests and the expected response time of class NC requests, respectively, both as a function of number of mini-cycles, i.e., N_{mc}. These figures illustrate interest-

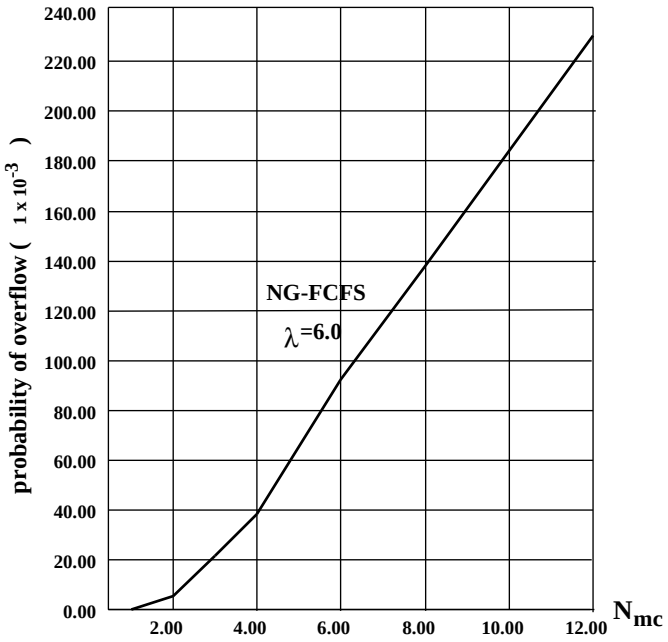

Fig. 8. Probability of overflow of class C requests as a function of number of mini-cycles (N_{mc}).

ing design tradeoffs, which were described in Section 2. Specifically, increasing the number of mini-cycles shortens their length and reduces the response time of class NC requests through more frequent opportunities for service. However, this technique results in diminishing returns beyond a certain number of mini-cycles. At the same time, higher values of N_{mc} result in fewer opportunities for seek optimization for class C requests. This in turn results in a higher overall class C load on the system, which contributes to two effects: less time in the cycle left for service of class NC requests as well as higher probability of overflow for class C requests. Due to these opposing effects, there exists an optimum number of mini-cycles which minimizes the mean response time of class NC customers. In the example of Figure 9 this value is $N_{mc} = 4$. Of course, an important consideration here is whether the required QoS characteristics for class C requests (i.e., $p = 0.01$) can be sustained at $N_{mc} = 4$, which is not the case in the example of Figure 8. Thus, our models can be used, for instance, to choose an appropriate number of mini-cycles so as to satisfy the performance and QoS requirements of both classes of customers.

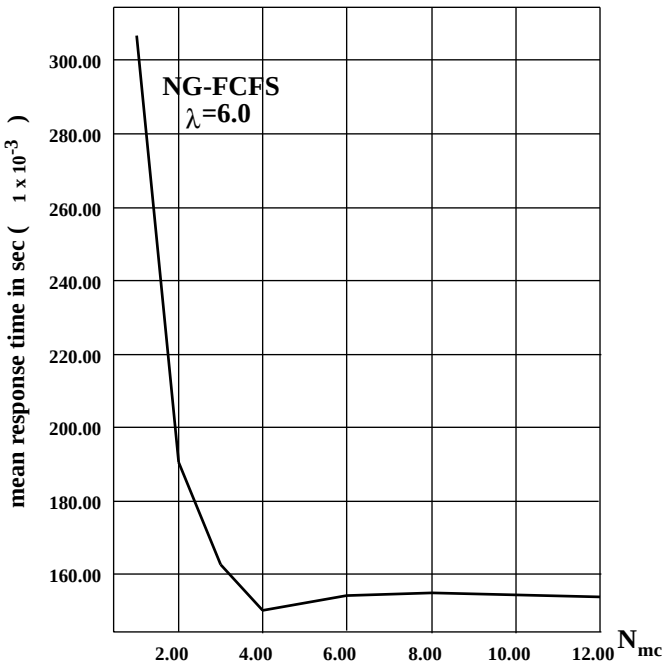

Fig. 9. Expected response time of class NC requests as a function of number of mini-cycles (N_{mc}).

5 Conclusions

Motivated by the need for mixed workload storage servers as well as the need for performance studies that can lead to better designs of such servers, in this paper we presented a family of scheduling algorithm for mixed workload storage servers. We then developed a set of corresponding *analytical* non-Markovian models with *tractable* analytical solutions. We showed that our solution methodology is flexible. It applies to the *entire family* of models, and it can be used to obtain performance measures of interest for both continuous and non-continuous classes of customers. Future work includes: (a) further performance studies of mixed workload systems using our analytical models and corresponding solution techniques; (b) improvements in computational complexity of the solution technique, e.g., by considering special structure that is present in these models, in order to facilitate studies of larger systems.

References

[1] S. Berson, S. Ghandeharizadeh, and R. Muntz. Staggered Striping in Multimedia Information Systems. In *Proc. of the ACM SIGMOD Conf. 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] R.M.L.R. Carmo, L.R. de Carvalho, E. de Souza e Silva, M.C. Diniz, and R.R. Muntz. Performance/availability modeling with the TANGRAM-II modeling environment. *Performance Evaluation*, 33:45–65, 1998.

[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] E. de Souza e Silva and H. R. Gail. Calculating cumulative operational time distributions of repairable computer systems. *IEEE Transactions on Computers (Special Issue on Fault Tolerant Computing)*, C-35:322–332, 1986.

[6] E. de Souza e Silva, H.R. Gail, L. Golubchik, and John C.S. Lui. Analytical models for mixed workload multimedia storage servers. Technical Report CS-TR-98-08, The Chinese University of Hong Kong, December, 1998.

[7] Edmundo de Souza e Silva and H. Richard Gail. Analyzing Scheduled Maintenance Policies for Repairable Computer Systems. *IEEE Transactions on Computers*, 39(11):1309–1324, Nov. 1990.

[8] Edmundo de Souza e Silva, H. Richard Gail, and Richard R. Muntz. Efficient Solutions for a Class of Non-Markovian Models. *Computations with Markov Chains: Proceedings of the 2nd Intl. Workshop on the Numerical Solution of Markov Chains, Edited by W. J. Stewart*, pages 483–506.

[9] Edmundo de Souza e Silva, H. Richard Gail, and Richard R. Muntz. Polling Systems with Server Timeouts and Their Application to Token Passing Networks. *IEEE/ACM Transactions on Networking*, 3(5):560–575, October 1995.

[10] S. Ghandeharizadeh and R. R. Muntz. Design and implementation of scalable continuous media servers. *Parallel Computing Journal, special issue on Parallel Data Servers and Applications*, 24(1):123–155, 1998.

[11] L. Golubchik, J. C.S. Lui, E. de Souza e Silva, and H. R. Gail. Evaluation of tradeoffs in resource management techniques for multimedia storage servers. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, June, 1999.

[12] W. K. Grassmann. Finding transient solutions in Markovian event systems through randomization. In *W. J. Stewart, editor, Numerical Solutions of Markov Chains*, pages 357–371. Marcel Dekker, Inc., 1991.

[13] A. Jensen. Markoff chains as an aid in the study of Markoff processes. *Skandinavsk Aktuarietidskrift*, 36:87–91, 1953.

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

[15] R. Nelson. *Probability, Stochastic Processes, and Queueing Theory : The Mathematics of Computer Performance Modelling.* Springer Verlag, 1995.

[16] 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.

[17] 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.

[18] G. Nerjes, P. Muth, and G. Weikum. Stochastic Service Guarantees for Continuous Data on Multi-Zone Disks. In *Proc. of the 16th Symp. on Principles of Database Systems (PODS'97)*, May 1997.

[19] G. Nerjes, Y. Romboyannakis, P. Muth, M Paterakis, P. Triantafillou, and G. Weikum. On Mixed-Workload Multimedia Storage Servers with Guaranteed Performance and Service Quality. In *Proc. of the 3rd Intl. Workshop on Multimedia Information Systems*, Sept. 1997.

[20] B. Ozden, A. Biliris, R. Rastogi, and A. Silberschatz. A Low-Cost Storage Server for Movie on Demand Databases. *Proc. of the 20th Intl. Conf. on Very Large Data Bases*, Sept. 1994.

[21] 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.

[22] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, 1994.

[23] 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.

[24] 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.

[25] Z.-L. Zhang, J. Kurose, J. Salehi, and D. Towsley. Smoothing, Statistical Multiplexing and Call Admission Control for Stored Video. In *IEEE JSAC*, 1997.