Stochastic Performance Guarantees for Mixed Workloads in a Multimedia Information System *

Guido Nerjes[†], Peter Muth[‡], Gerhard Weikum[‡]

[‡]University of the Saarland
Department of Computer Science
D-66041 Saarbrücken, Germany

[†]Swiss Federal Institute of Technology (ETH)
Institute of Information Systems
CH-8092 Zurich, Switzerland

E-mail: {nerjes, muth, weikum}@cs.uni-sb.de, WWW: http://www-dbs.cs.uni-sb.de/

Abstract

We present an approach to stochastic performance guarantees for multimedia servers with mixed workloads. Advanced multimedia applications such as digital libraries or teleteaching exhibit a mixed workload with accesses to both 'continuous' and conventional, 'discrete' data, where the fractions of continuous-data and discrete-data requests vary over time. We assume that a server shares all disks among continuous and discrete data, and we develop a stochastic performance model for the resulting mixed workload, using a combination of analytic and simulation-based modeling. Based on this model we devise a round-based scheduling scheme with stochastic performance guarantees: for continous-data requests, we bound the probability that 'glitches' occur, and for discrete-data requests, we bound the probability that the response time exceeds a certain tolerance threshold. We present early results of simulation studies.

1 Introduction

Multimedia applications such as news-on-demand for the home market or digital library access over the Internet pose challenging performance demands on the underlying storage servers [14, 7]. Previous research in this area has very much focused on the performance issues for *continuous data* only (i.e., video and audio), assuming that all client requests refer to movies or video clips. However, advanced applications such as digital libraries or teleteaching will rather exhibit a *mixed workload* with massive access to conventional, *discrete data* such as HTML text documents and images as well as index-supported searching in addition to the requests for continuous data. Furthermore, with unrestricted 24-hour world-wide access over the Web, such multimedia servers have to cope with a dynamically evolving workload where the fractions of continuous-data vs. discrete-data requests vary over time and cannot be reliably predicted in advance. Thus, for a good price/performance ratio it is mandatory that such a server operates with a *shared resource pool* rather than statically partitioning all resources (disks, memory, etc.) into two pools for continuous and discrete data.

This paper addresses some of the critical performance issues that arise in the disk scheduling for mixed workload servers with a shared disk pool. A well known requirement is that continuous data calls for *performance guarantees* in terms of data delivery time to ensure 'hiccup-free' display at the client site. In addition, quality-conscious applications require that the response time of discrete-data requests stay below some user-tolerance threshold as well. This requirement has been ignored in prior work on mixed multimedia workloads [25] where the performance of discrete-data requests seems to be an afterthought. In fact, given that video viewing is commonly perceived as a higher-class service compared to the bread-and-butter access to 'standard' discrete data, it is likely that users will accept an admission control policy for resource-intensive continuous-data requests and may be turned away when the system saturates, whereas slow service for the vanilla requests to discrete data would be considered as unacceptable. Admission control for discrete-data requests, on the other hand, is out of the question from the user's viewpoint (although some WWW servers resort to this solution upon high load, but this is exactly perceived as non-responsive service).

Research on multimedia storage systems has put emphasis on deterministic worst-case guarantees for continuous data. Worst-case guarantees may indeed be reasonable in specific video-only applications such as movie-on-demand or in extremely critical real-time applications such as telesurgery. However, in the more general setting considered here, deterministic worst-case guarantees are infeasible and we argue that *stochastic guarantees* are more appropriate for the following reasons:

- The load imposed by the discrete-data requests can be characterized only in a stochastic manner (typically as a Poisson arrival process with a certain probability distribution for the service demands). In more technical terms [18], this fraction of the load is better captured by an open system model whereas the continuous-data requests can indeed be adequately modeled as a closed system with

* *This work has been supported by the ESPRIT LTR project HERMES.*

In Proceedings of the IEEE International Workshop on Research Issues in Data Engineering (RIDE'97), Birmingham, UK, April 1997

a fixed multiprogramming level and periodic service demands.

- The resource demands of both discrete- and continuous-data requests cannot be realistically modeled in a deterministic manner. Important performance factors like disk controller caches or masking of transient failures (e.g., in a RAID) can be captured only stochastically at best. In order to employ a deterministic model, one would have to assume very conservative service-time bounds (e.g., with no disk caching at all) so that scheduling policies may end up with substantially underutilized resources. Alternatively, a deterministic model could be based on mean values only (e.g., average disk seek time), but then it is impossible to give hard performance guarantees.

In this paper, we will therefore pursue stochastic guarantees for both continuous- and discrete-data requests. For discrete-data requests we aim to guarantee good response time with high probability by bounding the tail of their response time distribution (e.g., the 95th percentile). For continuous-data requests, on the other hand, we aim to bound the probability that data portions are behind their delivery deadline according to the real-time display requirements. Our approach is based on coarse-grained striping of the data across the server's disks and a disk scheduling scheme that operates in *rounds* similar to the schemes in [2, 6, 11, 23]. Each round (of say a few seconds duration) is divided into two periods, a *C-period* and a *D-period*, during which the continuous-data and the discrete-data requests are served. The length ratio of the two periods is a degree of freedom that is dynamically adapted to the current workload based on a prediction of the near-future response time of the discrete-data requests. Whenever the specified response time guarantees (in the order of a few seconds) can no longer be satisfied, it is attempted to shorten the length of the C-period and extend the D-period correspondingly. This in turns triggers an admission control and scheduling problem for the continuous-data requests within the C-period of a round.

Our approach uses an analytic model for stochastically estimating the 'glitch' rate, i.e., probability that a data portion does not meet its display deadline, for a given multiprogramming level of continuous-data streams between clients and the server. When a request arrives to open a new continuous-data stream, it is admitted only if the predicted glitch rate for the new multiprogramming level does not exceed a specified tolerance threshold. To estimate the response time distribution of the discrete-data requests, we investigate the use of analytic queueing models, specifically M/G/1 vacation server models. However, as it turns out that we can (so far) not derive a model that is both accurate and computationally inexpensive, we finally resort to using a carefully constructed simulation model that is evaluated off-line and can efficiently drive the system's run-time decisions.

We are not aware of any previous work that pursues an approach to mixed workloads along these lines. A prototype system based on this new approach is being built by extending the FIVE experimental file system [29, 30, 32].

The rest of the paper is organized as follows. Section 2 introduces our system architecture. Section 3 develops an approach towards a performance prediction model, for both continuous-data and discrete-data requests. Finally, Section 4 discusses the admission control and the adaptation of the disk scheduling to the workload dynamics, based on precomputed results of the performance model.

2 System Architecture

In this section, we discuss assumptions on the workload and the architecture of our system. In general, we assume that clients submit requests for both *continuous data* and *discrete data*, in short: *C-data* and *D-data*, to the server. Objects consisting of C-data, e.g., videos, audios or animation data, are composed of sequences of *fragments* and constitute data *streams* that are consumed by the client in a time-constrained way according to the display bandwidth of the object. In contrast, D-data such as text, images, or metadata, has no explicit time constraints. For C-data requests, in short *C-requests*[1], we assume that each client provides a certain amount of memory as a buffer for incoming fragments. The buffer size may vary among clients according to the local resources available. The buffer size must not be below a certain minimum allowing the server to deliver a fragment before the previous one is consumed by the client. We assume a fast and reliable network with a performance capacity well above our bandwidth requirements, and thus disregard network issues in this paper.

2.1 Data Layout

We consider a single server with D disks. Since video and audio compression techniques reduce the bandwidth of videos and audios substantially, we assume that the display bandwidth $r_{display}$ of a continuous object is always smaller than the bandwidth r_{disk} of a single disk.

C-data is split into fragments. Fragments are assigned to disks in a round robin fashion, similarly to the coarse-grained striping approach of [23] and the simple/staggered striping approach of [2] specialized to the case with cluster size 1 and stride 1. The salient properties that we share with these approaches are twofold:

- The load is balanced across the disks, assuming that continuous objects are sufficiently large to be spread across all disks and that most users consume complete objects (as opposed to fast-forwarding a video or viewing only a short prefix).
- The server can sustain more concurrent (but time-wise unrelated) streams on the same continuous object than it

1. Likewise, requests to D-data will henceforth be called *D-requests.*

would be possible by multiplexing the service of a single disk in case the entire object resided on this disk. With D disks, disk bandwidth r_{disk}, and object display bandwidth $r_{display}$, the server can, in principle, support up to $D * r_{disk} / r_{display}$ streams on the same or different objects (under the optimistic assumption that multiplexing does not lead to a reduction of the effective disk bandwidth).

We do not assume the display bandwidth of a continuous object to be constant, as compression techniques such as MPEG-2 result in a variable bandwidth over time. In our scheme, all data fragments stored by the server have the same display time [9, 3], i.e., the time it takes a client to consume a fragment (e.g., a few seconds). As a consequence, fragments vary in size. By 'normalizing' all fragments to the same time length, we induce a periodic access pattern with a uniform period across all continuous objects regardless of the display bandwidth differences between objects and the variation within an object. This type of fragmentation requires parsing a continuous object before it is laid out on the server's disks, but this is straightforward and inexpensive given that continuous objects are never modified after their initial insertion.

Discrete objects are allocated to disk such that the expected I/O load on behalf of this data is balanced across the disks; this involves coarse-grained striping for large objects and simple but effective load balancing heuristics along the lines of [30]. In what follows we do, however, not rely on any specific assumptions on the storage layout for discrete objects.

C-data and D-data both reside on the same shared disk pool, as this provides a much better resource utilization than a partitioned scheme with dedicated disks, from both a disk space and a disk bandwidth point of view:

- For a partitioned server, new disks have to be added when the space for one of the two data categories becomes exhausted. A server with shared disks requires additional disks only if it runs out of space for both kinds of data together. The difference becomes important when the ratio of space used by C-data and D-data varies over time. This is the case, for example, for many Web servers with evolving sets of HTML documents.
- The advantage of a shared disk pool is even more important from a disk bandwidth viewpoint. For many multimedia applications it is likely that the ratio of C-requests and D-requests varies over time. For example, teleteaching lectures in the morning may be mostly based on videos, whereas working on assignments in the afternoon requires more lookups of reference data such as text. A partitioned server needs enough disk performance capacity on each data type in order to sustain all workloads. On a shared server, the aggregate bandwidth of all disks can be used anytime to serve both C- and D-requests. Thus, sharing disks may yield a substantial cost saving.

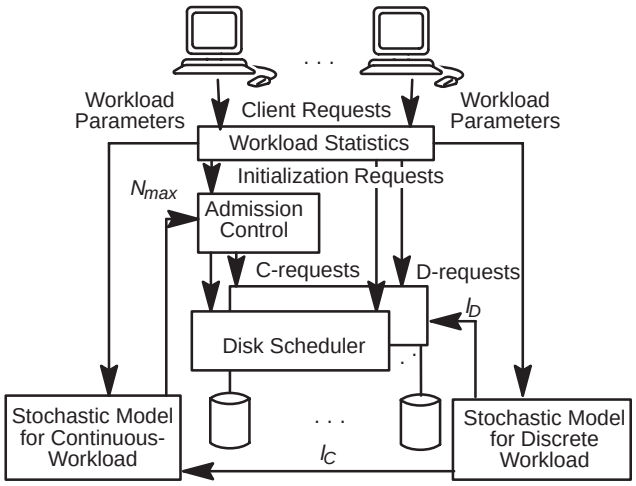

Figure 1: Architecture of the Mixed Workload Server

2.2 Admission Control and Disk Scheduling

The scheduling is composed of an admission control and a separate disk scheduler for each disk, as illustrated in Figure 1. D-requests are immediately submitted to the responsible disk schedulers for those disks on which the requested object resides. For simpler explanation, we will assume that all D-requests are served by exactly one disk as it would indeed be the case with non-striped objects. *Initialization requests* for opening a new continuous data stream have to pass the admission control first. As C-requests are deadline-oriented, only a limited number of concurrent streams can be sustained. Therefore, the admission control rejects new initialization requests when the server load becomes too high (as computed by the stochastic model of Section 3.1.). D-requests are not subject to admission control because we assume that this is not acceptable to the applications that consider such requests as a bread-and-butter workload. If D-requests were rejected by some form of admission control, they would have to be queued until they are eventually accepted. From the application point of view, the queueing time just adds to the response time perceived by the user.

Now consider the actual disk scheduling. The periodic pattern of the C-requests for the admitted streams suggests a cyclic scheduling scheme that proceeds in rounds, with a round length, l_{round}, equal to the display time of a fragment, which is uniform across all fragments. The round length is a configuration parameter of our architecture; changing it would require all C-data to be re-fragmented. During a round, all C-requests of the admitted streams have to be served. Given that a fragment always resides on a single disk, there are no dependencies among the requests of one round, so that we can schedule the requests of each disk separately, as long as we complete all requests by the end of the round. In order to minimize disk seeks, we use the SCAN algorithm for the disk arm movement [28, 4] (also known as the 'elevator'

algorithm). With this algorithm, all requests of one round are sorted according to their seek position on the disk and are served with one sweep of the disk arm.

So far we have disregarded the scheduling of D-requests. Since the round length would be at most a few seconds, which is a tolerable delay for most applications, an intriguing idea would be to schedule the D-requests after all C-requests of a round are served. However, since this unused time at the end of a round can become arbitrarily small depending on the C-request load, the response time of D-requests may degrade significantly. Discrete requests may end up being queued for several rounds, unless the scheduler takes additional actions. To avoid this situation, we further refine the approach and divide a disk scheduling round into two periods of pre-specified length: In the first period, denoted *C-period*, C-requests are served by the SCAN algorithm. The second period, denoted *D-period*, is used to serve D-requests, with an FCFS policy for fairness reasons. The ratio l_C/l_D of the lengths of both periods is a scheduling parameter and has to be adjusted to reflect changes in the overall workload. A longer *C-period* is needed if a higher number of continuous data streams must be sustained, and a longer *D-period* is needed when the arrival rate of D-requests increases. As the overall round length $l_{round} = l_C + l_D$ is constant, it is impossible to support both request categories in an optimal way. Our approach to finding an adequate compromise is based on two stochastic model components, as shown in Figure 1:

(1) For the discrete requests, based on the assumption of a Poisson arrival process with a given arrival rate λ_D and a certain, observable distribution of the service time, the length l_D is derived such that the response time for the majority of discrete requests, say the 95th percentile, is below a certain threshold (which would typically be in the order of a few seconds).

(2) Then, given the length $l_C = l_{round} - l_D$ of the C-period, we derive the maximum number, N_{max}, of concurrent C-data streams that can be served during a C-period such that the probability, p_{late}, of missing a display deadline stays below a specified threshold, say 99 percent. The computed value of N_{max} is then used to drive the admission control in that only up to N_{max} streams can be admitted.

Using both models gives us stochastic guarantees for C-requests not missing their deadlines and for discrete request not exceeding a given response time threshold. The stochastic models themselves are discussed in the next section.

3 Performance Guarantees

This section develops stochastic models that allow us to give stochastic guarantees for the glitch rate of C-requests and the response time of D-requests. These guarantees are derived as Chernoff bounds [19, 22] for the tail of the underlying probability distributions. Throughout the section we consider only one disk and its corresponding load, which is feasible as there are no scheduling dependencies among different disks (see Section 2). Thus, all workload parameters like the multiprogramming level of continuous streams and the arrival rate of D-requests are on a per disk basis, assuming that the load is uniformly distributed across disks.

3.1 Performance Guarantees for C-Requests

The goal of this section is to derive, for a C-period of length l_C, an upper bound, N_{max}, for the number N of concurrent streams such that the probability of not being able to serve all N requests within time l_C is below a certain threshold p_{late}, say 1 percent. From this probability, p_{late}, we derive the probability mass function f_{glitch} for the number of 'glitches', i.e., late data deliveries, within an entire stream of duration n_R rounds as follows:

$$f_{glitch}(k) = P[\#\ of\ glitches{=}k] = \binom{n_R}{k}\ p_{late}^{\ k}\ (1 - p_{late})^{n_R - k}$$

Note that this is actually a pessimistic upper bound for the glitch rate of an individual stream, as a late round would affect only a subset of the active streams. We will later use these probabilistic considerations in the admission control for newly arriving initialization requests (see Section 4), with a specified bound on p_{late} or, equivalently, the tail of f_{glitch}.

The key problem to be solved here is to estimate the total service time for the N requests of one round's C-period, using a SCAN policy for the disk arm movement. Prior work on this problem used constant worst case values for the seek and rotational delays between successive data transfers [12], or assumed that the total (i.e., accumulated) seek time of one sweep over the disk equals the maximum seek time of the disk [23]. This yields a deterministic but unrealistic estimate since it ignores the stochastic nature of rotational delays and the non-linearity of the disk arm movement [26]. The only work that addresses this problem by a stochastic model are [5, 8]. [5] assumes independent seeks for the N requests rather than the much better SCAN policy, and arrives at a relatively coarse bound based on the Tschebyscheff inequality. [8] is also based on independent seeks and assumes that N is sufficiently large to apply the central limit theorem (i.e., consider only the limit $N \to \infty$) and thus assume that the total service time is normally distributed, which is not always justified for realistic values of N (e.g., 10 to 50 streams per disk). In the following we derive a much more accurate stochastic model and a much tighter bound using a recent result on the total seek time of the SCAN policy [24] and the method of Chernoff bounds [19, 22].

Let T_C denote the total service time for a C-period with N requests. Then we have

$$T_C = T_{seek} + \sum_{i=1}^{N} T_{rot,i} + \sum_{i=1}^{N} T_{trans,i}$$

where T_{seek} is the accumulated seek time for one sweep of the SCAN policy, $T_{rot,i}$ is the rotational delay and $T_{trans,i}$ is the transfer time of the ith request.

According to [24] T_{seek} is maximized, under a realistic function for the seek time, for equidistant seek positions of the N requests. The seek time function itself is assumed to be proportional to the square root of the seek distance for small distances below a disk-specific constant, and a linear function of the seek distance for longer distances, which is in accordance with the studies of [26]. Thus, for given disk parameters, the maximum total seek time of a sweep can be easily computed by assuming the N seek positions to be at cylinders $i*CYL/(N+1)$ for $i=1, ..., N$ where CYL is the total number of the disk's cylinders, and applying the seek time function. This computation yields an upper bound for T_{seek} which, other than depending on N, can now be viewed as a constant, denoted $SEEK$ in the following.

The N random variables $T_{rot,i}$ are independently and identically distributed with a uniform distribution between 0 and the time for one disk revolution, ROT. Similarly, the random variables $T_{trans,i}$ are independently identically distributed. This distribution depends on the distribution of data fragments and the disk's transfer rate (which in turn is a function of the revolution speed and the head switch time). For the sake of a simpler explanation, we assume in the following that $T_{trans,i}$ is exponentially distributed with a mean value $TRANS$. (The same derivation could be carried out also with other common distributions, e.g., a more realistic hyperexponential distribution, but this would complicate the formulas.)

So T_{seek} is equal to the constant $SEEK$, and the probability density functions of $T_{rot,i}$ and $T_{trans,i}$ are given by

$$f_{rot}(x) = \frac{1}{ROT} \quad \text{and} \quad f_{trans}(x) = \frac{1}{TRANS}\ e^{-\frac{x}{TRANS}},$$

and their Laplace-Stieltjes transforms [1, 22] are given by

$$L_{seek}(s) = e^{-s\,SEEK},\ L_{rot}(s) = \frac{1 - e^{-s\,ROT}}{s\ ROT}, \text{ and}$$
$$L_{trans}(s) = \frac{1}{1 + s\,TRANS}.$$

The Laplace transform of T_C, which involves the N-fold convolution of the convolution of $T_{rot,i}$ and $T_{trans,i}$, is given by

$$L_C(s) = e^{-s\,SEEK} \left(\frac{1 - e^{-s\,ROT}}{s\ ROT}\right)^N \left(\frac{1}{1 + s\,TRANS}\right)^N,$$

and the corresponding moment generating function $M_C(s)$ equals $L_C(-s)$. Now we are ready to apply Chernoff's theorem to bound the tail of the random variable T_C. Namely, the following inequation holds [19, 22]:

$$P[T_C \geq t] \leq inf_{\theta \geq 0} \{e^{-\theta t}\ M_C(\theta)\} = inf_{\theta \geq 0} \{g(\theta)\}$$

with

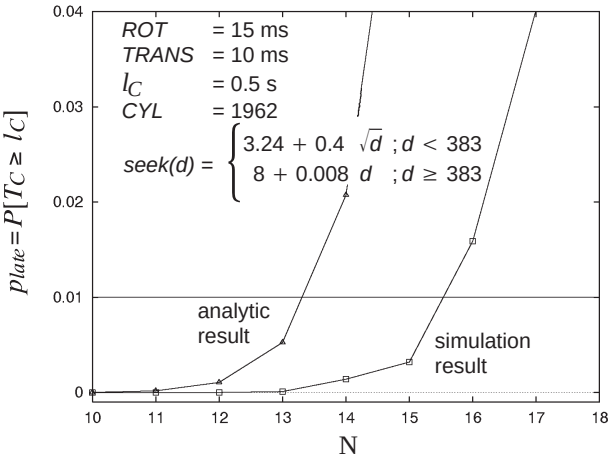

Figure 2: Analytically Predicted vs. Simulated 'Lateness' Distribution

$$g(\theta) = e^{-\theta t}\ e^{\theta\,SEEK} \left(\frac{e^{\theta\,ROT} - 1}{\theta\ ROT}\right)^N \left(\frac{1}{1 - \theta\,TRANS}\right)^N.$$

For the given form of g, differentiating g and solving $g' = 0$ for θ yields the optimum value of θ to obtain the sharpest bound in the Chernoff inequation. While we did not manage to obtain a closed form expression for this result, solving $g' = 0$ numerically is straightforward and very efficient.

So finally p_{late} is obtained by

$$p_{late} = P[T_C \geq l_C] \leq g(solution\ of\ g' = 0\ using\ t = l_C)$$

For example, for $l_C = 0.542$, $ROT = 0.015$, $TRANS = 0.01$, $SEEK = 0.12282$, $N = 15$, the derived upper bound for p_{late} is approximately 0.01 . In other words, we can guarantee with probability at least $1 - p_{late} = 0.99$ that all N C-requests of one round can be served within the C-period of length l_C. So for a given value of l_C and a threshold δ for p_{late}, we can derive the maximum number of concurrent streams as $N_{\max} = \max\{N \mid p_{late} \leq \delta\}$. For example, if $l_C = 0.2$ and p_{late} should be at most 1 percent, then N_{max} would be 3, and for $l_C = 0.4$, N_{max} would be 9.

We compared the predictions of this model with results obtained from detailed simulations. Figure 2 shows the analytically predicted and the simulated values for p_{late} as a function of N_{max}. The analytic model is conservative in that it always overestimates the lateness probability. For example, for a p_{late} threshold of 0.01, the analytic model allows a maximum of 13 streams whereas the simulation shows that a value of 15 for N_{max} is possible.

Now that we have the probability p_{late} for not being able to complete all N requests within time l_C, we can apply the method of Chernoff bounds also to the probability mass function f_{glitch} of the number of glitches within an entire stream of n_R rounds. As derived above, the number of

glitches is binomially distributed. For this important case, the following Chernoff bound, derived in [HR89], can be applied to our scenario under the constraint that k / n_R > p_{late} :

$$p_{error} = P[\#\ of\ glitches \geq k]$$

$$\leq \left(\frac{n_R\, p_{late}}{k}\right)^{k} \left(\frac{n_R - n_R\, p_{late}}{n_R - k}\right)^{n_R - k}$$

For example, using p_{late} = 0.01 and n_R = 1200, the probability p_{error} for more than 24 glitches is at most 1 percent. Conversely, for a specified upper bound for the number of glitches per stream (the *k* value above) and the corresponding probability that this bound is not exceeded, we can again derive the maximum number of streams, N_{max}, that the system can sustain within a C-period of length l_C under these conditions.

For example, for l_C = 0.5, n_R = 1200, *k* = 12 and the other parameters as given in figure 2, simulations show that at most 17 streams can be accepted. The analytic evaluation restricts N_{max} to a maximum value of 12.

3.2 Performance Guarantees for D-Requests

3.2.1 Applicability of M/G/1 Vacation Server Models

In this section we explore the use of M/G/1 vacation server models for predicting the response distribution of the D-requests. In applications with a large number of clients, the arrival of requests can be stochastically described as a Poisson process. As a consequence, the time between the arrival of two successive D-requests is exponentially distributed with mean 1 / λ_D, where λ_D is the average arrival rate of the Poisson process. Because of this stochastic behavior it is not feasible to provide 'hard', deterministic performance guarantees to every single request (e.g. every request is served within 0.1 seconds). Rather it is common (see, e.g., database and transaction system benchmarks [13]) to require that a specified percentage of requests will finish within a given time, e.g., 95 percent of the D-requests will have a response time below 1 second.

As described in Section 2 the service of requests is performed in a cyclic manner. During each round of constant length l_{round} a D-period of length l_D is dedicated to the service of D-requests. D-requests that arrive outside the D-period are queued and served in FCFS order in the subsequent D-periods. This situation is illustrated in Figure 3. The performance measure that we aim to predict in this section is the response time of D-requests, which depends on the arrival rate λ_D, the length of the D-period l_D = l_{round} - l_C, and the service time distribution of the D-requests.

An analytic approach to the estimation of the response time of D-requests is to build on *vacation-server queueing models* [31, 10]. In an M/G/1 queueing model with vacations the arriving requests form a Poisson process with exponentially distributed interarrival time, the service time can be arbitrarily distributed, and the requests are served by a single server in FCFS order. The speciality of vacation-server models is that the server occasionally suspends its service for a certain 'vacation period'; in our setting the 'vacation' corresponds to the C-period during which the server pauses serving D-requests. In the following we discuss the suitability of specific vacation-server models for which analytic results can be found in the literature.

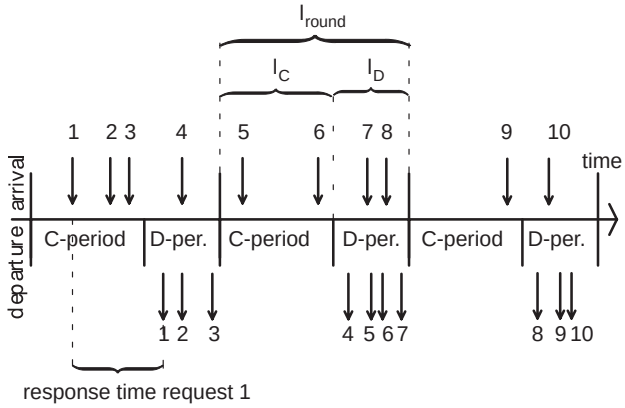

Figure 3: Arrivals and Departures of D-Requests

In vacation models with exhaustive service, the server takes a vacation (i.e., stops serving requests) for a certain time whenever its queue of waiting requests becomes empty. When the server returns from a vacation it takes another vacation (following a so-called 'multiple vacation model') if no request has arrived during the vacation and the queue is still empty [31]. A possible sequence of service and vacation periods is shown in Figure 4 (where service and vacation times are assumed to be constant).

Our cyclic service strategy of serving C-requests and D-requests can be coarsely mapped to the sketched M/G/1 vacation-server model with exhaustive service and multiple vacations: D-periods correspond to the service periods, C-periods to vacations of the server. Note that this is an approximation since our actual service policy does not take another vacation when the D-request queue is empty at the beginning of the D-period. Despite this discrepancy it seems intriguing

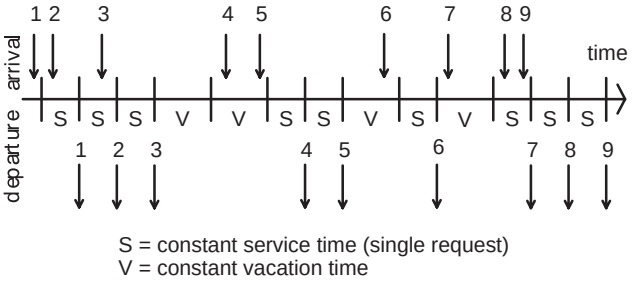

Figure 4: Service and Vacation Periods in the Vacation Server Model

to apply such a model for an approximate performance prediction as follows.

The response time of a D-request, T_{resp}, is composed of the time T_{wait} the request spends in the server queue until service starts and the time T_{svc} that is needed for service.

$$T_{resp} = T_{wait} + T_{svc}$$

For a simple presentation, it is assumed that the service time T_{svc} is exponentially distributed with mean $E[T_{svc}] = SVC$ (other distributions such as normal or hyper-exponential could be handled as well). The vacation time T_{vac} is constant with value $E[T_{vac}] = VAC = l_{round} - l_D$. The Laplace-Stieltjes transforms of the service time, vacation time, and waiting time [31] distributions are given by

$$L_{svc}(s) = \frac{1}{1 + s\,SVC},\ L_{vac}(s) = e^{-sVAC}, \text{ and}$$

$$L_{wait}(s) = \frac{1 - L_{vac}(s)}{s\,E[T_{vac}]} \cdot \frac{s\,(1 - \lambda\,E[T_{svc}])}{s - \lambda + \lambda\,L_{svc}(s)}.$$

The Laplace-Stieltjes transform of T_{resp} is then obtained as the product of the waiting and service time transforms:

$$L_{resp}(s) = L_{wait}(s) * L_{svc}(s) =$$

$$\frac{1 - e^{-sVAC}}{s\,VAC} \cdot \frac{s\,(1 - \lambda\,SVC)}{s - \lambda + \lambda\,(1 + s\,SVC)^{-1}} \cdot \frac{1}{1 + s\,SVC}$$

Now, analogously to Section 3.1., it is possible to apply Chernoff's theorem to bound the tail of the random variable T_{resp}. Let $M_{resp}(s) = L_{resp}(-s)$ be the moment generating function of T_{resp}. Then the following inequation holds:

$$P[T_{resp} \geq t] \leq inf_{\theta \geq 0} \left\{ e^{-\theta t}\ M_{resp}(\theta) \right\} = inf_{\theta \geq 0} \{ h(\theta) \}$$

with

$$h(\theta) = e^{-\theta t} \cdot \frac{e^{\theta VAC} - 1}{\theta\,VAC} \cdot \frac{\theta\,(1 - \lambda\,SVC)}{\theta + \lambda - \lambda\,(1 - \theta\,SVC)^{-1}} * \frac{1}{1 - \theta\,SVC}$$

Given a fixed response time threshold ρ, the probability that a request has a response time of at least ρ can be calculated using

$$p_{delay} = P[T_{resp} \geq \rho] \leq h(\textit{solution of } h' = 0 \textit{ with } t = \rho)$$

For example, for $SVC = 0.01$s, $VAC = 0.3$s, $\lambda = 30\text{s}^{-1}$ and $\rho = 0.338$ seconds, p_{delay} is bounded by 0.05. This means that with probability $1\text{-}p_{delay} = 0.95$ a request will finish within a time interval of 0.338 seconds. If, on the other hand, both the response threshold ρ and a bound δ for p_{delay} are given, then the maximum feasible vacation time T_{vac} or, equivalently, the minimum length of the D-period, $l_D = l_{round} - VAC$, can be determined for a certain arrival rate λ.

As noted before, the multiple vacation model with exhaustive service does not capture the periodic nature of our scheduling policy. Under light load, i.e., when the arrival rate is low and the waiting queue is often empty, the vacation model predicts more vacations than our policy which would actually take a vacation only once in a round. Under heavy load, i.e., when the waiting queue is rarely empty, the vacation model predicts only few vacations, which is in contrast to our actual policy where the server is definitely on vacation once in a round and the vacation length is not affected by the load. These discrepancies lead to deviations between the predicted performance as derived analytically and the results of a detailed simulation. Unfortunately, the large error we found renders the analytic model unacceptable for practical use.

Better approximations could be obtained by applying the more sophisticated models described in [20, 21]. These models assume that the length of a service period, in our case l_D, is limited. Whenever the server expires this limited service time, a vacation period is started regardless of whether the request queue is empty or not. In [20] the request in service is preempted when the service time expires and resumed directly after the vacation; in [21] the request in service is first completed before the vacation period starts. Unfortunately, however, these models incur substantial mathematical difficulties; there are no closed-formula results, and even approximative derivations involve a high computational complexity. Studies on improving the computational complexity of the solution methods of [20, 21] while retaining an acceptable accuracy are left for future work.

3.2.2 Simulation-Based Guarantees

Although the analytic model of Section 3.2.1 does provide better insight into the performance of D-requests, its inaccuracy, on the one hand, and the computational complexity of more accurate models, on the other hand, prevent us from adopting an analytic model for run-time scheduling decisions. So we rather resort to estimations based on off-line simulations. Assuming that the service time distribution, the round length l_{round}, and the user-tolerated response time percentile are fixed at the system configuration time, we can pre-compute by simulation the minimum value of l_D that is necessary to meet the requirements, over a spectrum of λ values. These values would then be stored in a table that can be efficiently looked up at run-time.

Although this approach may appear less elegant than an analytic model, it serves its purpose well in that it allows us to predict the response time distribution of the D-requests as a function of the two parameters l_D and λ. The reduction of the parameter space to these run-time-relevant parameters is what makes this approach feasible. Note that we can control the accuracy and statistical confidence of the predictions,

without incurring any additional run-time costs, as the simulations are carried out off-line.

4 The Scheduling Algorithm

Our disk scheduling algorithm is driven by the control parameters N_{max}, l_C, and l_D, where $l_C + l_D = l_{round}$ and l_{round} is fixed. In the previous section we have presented a stochastic model and a simulation approach to derive the values of l_C, l_D and N_{max} from given workload parameters:

(1) The arrival rate of D-requests, λ_D, and the specified threshold for the tail of the response time distribution determine the value of l_D and, with fixed l_{round}, also the value of l_C.

(2) The value of l_C and the specified threshold for the probability of missing a delivery deadline (or, equivalently, the glitch rate of a C-data stream) determine the maximum sustainable multiprogramming level of concurrent C-data streams, N_{max}.

These control parameters are adjusted whenever a change in the workload takes place. In principle, this can happen at the beginning of each round. But we expect that shifts in the workload are not that frequent, and the parameters remain stable over several rounds. The computation of the parameters is based on the analytic model of Section 3.1 and off-line simulations for the problem of Section 3.2. In both cases the results are precomputed and stored in tables that merely need to be looked up at run-time. These tables are very compact, as we can assume that service time distributions are stationary (as they are determined by disk and data properties) and thus do not have to be considered as variable parameters. Thus, the table for the l_D values has one entry for each value of λ_D within a range of expected values, say from 1 arrival per second through 10000, and could even be organized as a sparse table using interpolation for missing values. The table for N_{max} needs one entry for every possible setting of l_C and would be in the order of a thousand entries. Altogether, the run-time overhead of the approach is in the order of 10 KBytes of memory.

Figure 5 presents the complete admission control and disk scheduling algorithm in a pseudo-code notation. In addition to the tables discussed above, the main data structures are three queues, denoted *C-queue*, *D-queue*, and *I-queue*, for the C-requests, D-requests, and initialization requests, respectively. The algorithm is invoked at the beginning of each round, i.e., every l_{round} time units. The first step analyzes the current load conditions by inspecting the workload statistics and looking up the tables for the control parameters of the round. In the second step, it is attempted to admit new streams that may be waiting in the *I-queue*. However, for each disk, a limit of N_{max} streams must not be exceeded. Note that invoking the admission control at the start of each scheduling round implies an average startup latency for newly arriving initialization requests of half a round length. Given a typical round length of a few seconds, the startup delay appears to be tolerable.

```
step 1 (analysis):
    determine currently expected  l_D from
    workload statistics;
    look up the appropriate values of l_D, l_C, and N_max in
    the corresponding tables;
step 2 (admission control):
    let N be the current number of continuous requests
    that need to be served per round and per disk on
    behalf of the active streams;
    while (N <= Nmax) {
        admit first stream from the I-queue;
        insert the first request of each newly admitted
        stream into the C-queue;
        N++
    };
step 3 (overload management):
    if (N > N_max)
    case (policy)
        kill: kill N-N_max streams;
        adapt QoS: reduce quality of service of all
        streams by N/N_max;
        wait: delay adjustment of l_C until N-N_max streams
        have terminated;
step 4 (disk service):
    while (current_time < start_of_current_round + l_C) {
        process requests of the C-queue according to the
        SCAN policy;
    }
    while (current_time < start_of_current_round + l_round) {
        process next request from the D-queue
        in FCFS order;
    };
```

Figure 5: Pseudo Code for the Admission Control and Disk Scheduling Algorithm

The third step, overload management, is necessary because of the evolving load incurred by the D-requests. It may turn out that the C-round duration l_C has to be shortened in order to accommodate an increased discrete load. In such a situation, the system could have admitted already more streams than it can now sustain. The problem is how to drive the system back to a state where it is able to sustain the current load. There are at least three pragmatic options, iterated until, for each disk, the number of C-requests to be served in each round is again below N_{max}:

(1) Kill active streams

(2) Reduce the quality of service for active streams [16, 8], e.g., by dropping some video frames

(3) Keep the old length of the C-period until enough streams are finished

In cases (1) and (2), C-requests suffer from the increased load incurred by D-requests. In case (3), D-requests suffer until the load of C-requests is eventually reduced. Combinations of the above approaches are possible, but they remain pragmatic, as during the overload phase the performance guarantees are not met. In practice, this might be less of a

problem as in most cases the arrival rate λ_D of D-requests will change slowly, causing only slight changes in the scheduling parameters. However, we will address this problem in more detail in our future work.

The fourth and final scheduling step is the actual processing of the requests in the C- and D-queues. The processing of the C-queue should be finished l_C time units after the start of the round. C-requests that cannot be served until then cause a glitch in the corresponding data stream. If the last C-request to be served in the round finishes before the end of the C-period, the remaining time is dedicated to the D-queue, which makes the D-period longer.

5 Conclusions

In this paper, we have presented an approach towards stochastic performance guarantees for multimedia servers with workloads consisting of both continuous-data and discrete-data requests. This work is part of the Esprit long-term research project HERMES [15]. The architecture of our server in terms of data placement and load balancing is based on experiences with the FIVE prototype [29, 30, 32], an experimental file system for parallel disk systems. We are currently extending FIVE to support the presented admission control and scheduling method for both continuous and discrete data, using the stochastic model components developed in this paper. We plan to integrate the extended FIVE system with an already implemented prototype multimedia server for a 'News on Demand' application [27] (which is currently based on staggered striping).

Future work includes extensions of the architecture in order to make it more flexible. In particular, we have disregarded buffering issues so far, and we want to exploit caching opportunities especially at the client sites. In the advanced multimedia applications that we are aiming at, many clients are quite powerful PCs or workstations that have memory and also local disk resources that substantially exceed the minimum buffering capabilities of a client as opposed to a set-top unit in a home market setting. This allows the server to deviate from the usual just-in-time-delivery paradigm for the continuous data, and rather preload fragments into the client ahead of time depending on the client's available cache space, thereby saving resources for heavy-load periods later. On the other hand, with a more complex architecture, the complexity of the stochastic models increases, too. Therefore, the approach of pre-computing performance prediction results by off-line simulations (as pursued in Section 3.2) and using these results in an efficient table-lookup manner for run-time scheduling decisions may become more intriguing.

References

[1] Arnold O. Allen, *Probability, Statistics and Queueing Theory with Computer Science Applications*, 2nd edition, Academic Press, 1990.

[2] Steven Berson, Shahram Ghandeharizadeh, Richard Muntz, *Staggered Striping in Multimedia Information Systems.* Proceedings ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, pp.79–90, May 1994.

[3] Ariel Cohen, Walter A.Burkhard, P. Venkat Rangan, *Pipelined Disk Arrays for Digital Movie Retrieval*, Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS '95), Washington D.C., May 1995.

[4] Edward G. Coffman, Jr., Micha Hofri, *Queueing Models of Secondary Storage Devices*, In Hideaki Takagi, editor, Stochastic Analysis of Computer and Communication Systems, North Holland, 1990.

[5] Huang–Jen Chen, Thomas D. C. Little, *Storage Allocation Policies for Time–Dependent Multimedia Data*, to appear in IEEE Transactions on Knowledge and Data Engineering.

[6] Mon–Song Chen, Dilip D. Kandlur, Philip S. Yu, *Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogenous Multimedia Streams*, Proceedings of the ACM International Conference on Multimedia (ACM Multimedia '93), Anaheim, California, 1993.

[7] Stavros Christodoulakis, Peter Triantafillou, *Research and Development Issues for Large–Scale Multimedia Information Systems*. ACM Computing Surveys 27(4): pp. 576-579, 1995.

[8] Ed Chang, Avideh Zakhor, *Variable Bit Rate MPEG Video Storage on Parallel Disk Arrays*, Proceedings of SPIE Conference on Visual Communication and Image Processing, Chicago, Illinois, pp. 47-60, September 1994.

[9] Ed Chang, Avideh Zakhor, *Cost Analyses for VBR Video Servers*, Proceedings of IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology, San Jose, California, January 1996.

[10] Bharat Doshi, *Single Server Queues with Vacations*, In Hideaki Takagi, editor, Stochastic Analysis of Computer and Communication Systems, North Holland, 1990.

[11] D. James Gemmel, Jiawei Han, Richard Beaton, Stavros Christodoulakis, *Delay–Sensitive Multimedia on Disks*, IEEE Multimedia, pp. 57-67, 1995.

[12] Shahram Ghandeharizadeh, Seon Ho Kim, *Striping in Multi–disk Video Servers*, Proceedings of the SPIE High–Density Data Recording and Retrieval Technologies Conference, October 1995.

[13] Jim Gray (Ed.), *The Benchmark Handbook for Database and Transaction Processing*, 2nd edition, Morgan Kaufmann, San Mateo, 1993.

[14] D. James Gemmel, Harrick M. Vin, Dilip D. Kandlur, P. Venkat Rangan, Lawrence A. Rowe, *Multimedia Storage Servers : A Tutorial*, IEEE Computer, pp. 40-49, May 1995.

[15] Technical Reports ESPRIT Long Term Research Project Hermes (Foundations of High Performance Multimedia Information Management), No. 9141, [www.ced.tuc.gr/hermes/]

[16] Silvia Hollfelder, Achim Kraiß, Thomas C. Rakow, *A client–controlled Adaption Framework for Multimedia Systems*, Technical Report No. 1022, GMD–IPSI, Sankt Augustin, September 1996.

[17] Torben Hagerup, Christiane Rüb, *A Guided Tour of Chernoff Bounds*, Information Processing Letters 33, pp. 305-308, 1989.

[18] Raj Jain, *The Art of Computer Systems Performance Analysis*, Wiley, 1991.

[19] Leonard Kleinrock, *Queueing Systems, Volume 1: Theory*, Wiley, 1975.

[20] Kin K. Leung, Martin Eisenberg, *A single–server queue with vacations and non-gated time-limited service*, Performance Evaluation 12, pp. 115-125, 1991.

[21] Kin K. Leung, David M. Lucantoni, *Two vacation models for token-ring networks where service is controlled by timers*, Performance Evaluation 20, pp. 165-184, 1994.

[22] Randolph Nelson, *Probability, Stochastic Processes, and Queueing Theory : The Mathematics of Computer Performance Modeling*, Springer, 1995.

[23] Banu Özden, Rajeev Rastogi, Avi Silberschatz, *Disk Striping in Video Server Environments*, Proceedings IEEE International Conference on Multimedia Computing and Systems, June 1996.

[24] Yen–Jen Oyang, *A tight upper bound of the lumped disk seek time for the scan disk scheduling policy*, Information Processing Letters 54, pp. 355-358, 1995.

[25] A. L. N. Reddy, Jim Wyllie, *I/O issues in a multimedia system*, IEEE Computer, 27(3), pp. 69–74, March 1994.

[26] Chris Ruemmler, John Wilkes, *An Introduction to Disk Modelling*, IEEE Computer, 27(3), pp. 17-28, March 1994.

[27] Nivo Randriam, Ulrike Wolf, *A Multimedia Storage Server for a News Archive* (in German), Diploma Thesis, Department of Computer Science, University of the Saarland, Saarbrücken, 1996.

[28] Abraham Silberschatz, Peter Galvin, *Operating System Concepts*, 4th edition. Addison–Wesley, New York, 1994.

[29] Peter Scheuermann, Gerhard Weikum, Peter Zabback, *Disk Cooling in Parallel Disk Systems*, IEEE Data Engineering Bulletin Vol.17 No.3, pp. 29–40, September 1994.

[30] Peter Scheuermann, Gerhard Weikum, Peter Zabback, *Data Partitioning and Load Balancing in Parallel Disk Systems*, Technical Report A/02/96, Department of Computer Science, University of the Saarland, 1996, submitted for publication.

[31] Hideaki Takagi, *Queueing Analysis : A Foundation of Performance Analysis, Volume 1 : Vacation and Priority Systems*, North Holland, Amsterdam 1991.

[32] Peter Zabback, *I/O Parallelism in Database Systems – Design, Implementation and Evaluation of a Storage System for Parallel Disks* (in German), Doctoral Thesis, Department of Computer Science ETH Zurich, 1994.