Rotational-Position-Aware Real-Time Disk Scheduling Using a Dynamic Active Subset (DAS)

Lars Reuther Martin Pohlack
Dresden University of Technology
Department of Computer Science
01062 Dresden, Germany
{reuther, pohlack}@os.inf.tu-dresden.de

Abstract

Scheduling disk requests with service guarantees has to bring the demand to meet guarantees in line with the need to optimize disk utilization. This paper presents the design, implementation and experimental evaluation of a disk-scheduling framework which optimizes the disk utilization under the constraints of both hard and statistical service guarantees. The framework is based on two principles: 1) upon each scheduling decision, the calculation of a subset of the outstanding disk requests such that all service guarantees can be enforced under worst-case assumptions and 2) the scheduling of this subset based on the rotational position of requests in order to improve the disk utilization. Results are presented through an implementation of the scheduling framework in DROPS, the Dresden Real-Time Operating System.

1. Introduction

Disk-storage usage in modern systems includes traditional, best-effort file access as well as the storage and retrieval of video and audio streams. The latter application constrains the ability of a disk-request scheduler to optimize the disk utilization. While best-effort applications have relatively weak requirements, like "good" response times, the retrieval of a video stream requires that deadlines for individual disk requests are met. A disk-request scheduler has to bring these constraints in line with the need to optimize the disk utilization which is crucial for good overall performance of the storage system.

Giving service guarantees for disk requests is a challenging requirement for a disk-request scheduler. Because of the physical design of disk drives, the execution times of disk requests have a poor ratio of average and worst-case execution times. However, the knowledge of the execution-time distributions can be used to give statistical service guarantees. If an application can tolerate occasional deadline misses, statistical service guarantees can be used to substantially improve the disk utilization compared to hard service guarantees [4].

This paper introduces and evaluates an admission-control and scheduling algorithm which (a) can provide both hard and statistical service guarantees for disk requests and (b) uses an SATF (*shortest access time first*) based request scheduler to optimize the disk utilization. Service guarantees are ensured by calculating the active subset of the outstanding disk requests upon each scheduling decision such that no guarantees will be violated regardless of which request of this subset is executed. Then, this subset is scheduled based on the rotational position of the disk requests.

The rest of this paper is organized as follows: The next section gives an overview of a general admission-control algorithm which can give both hard and statistical real-time guarantees. In Section 3, we describe the application of that admission-control algorithm to disk storage and a disk-scheduling algorithm which enforces service guarantees as well as optimizes the disk utilization. In Section 4, we evaluate the implementation of our scheduler. Section 5 gives an overview of related work and Section 6 concludes the paper.

2. Background: Quality-Assuring Scheduling in DROPS

The interest in statistical guarantees in real-time systems has been sparked by the observation that certain types of applications (most notably multimedia applications) cannot afford the poor resource utilization yielded by absolute guarantees, but at the same time, such applications can tolerate occasional deadline misses to a certain extent. The work we present in this paper was conducted in the context of *DROPS*, the *Dresden Real-Time Operating System* [5]. DROPS employs *Quality-Assuring Scheduling (QAS)* [4]

which aims to improve the resource utilization of real-time systems by:

- splitting calculations into one mandatory part and several optional parts. The mandatory part has to be executed under any circumstances, whereas optional parts may be dropped in case of resource shortage; and
- using a probabilistic model in the admission control which assures a certain quality parameter (i.e., the percentage of optional parts which meet their deadline).

The goal is to schedule active resources such that under overload the system behavior degrades gracefully and predictably. Gracefully means that only optional parts are omitted, and predictably means that the number of dropped parts can be predicted.

More precisely, a periodic task T_i is a sequence of jobs J_{ij}, where J_{ij} is to be executed in the j^{th} period of T_i. Each job J_{ij} consists of a mandatory part M_{ij} and one or more optional parts $O_{ij1}, O_{ij2}, ..., O_{ijc_i}$; the number c_i of optional parts is fixed for each task. Each of these parts is considered *successful* if it is completely executed. This condition must hold for all mandatory parts of a task, but it is sufficient that only a given percentage of optional parts is successful. To ensure that all optional parts reach the requested quality but do not consume more resources than necessary in case of resource shortage, a resource scheduler must enforce a *reservation time* r_i. The reservation time is a constant amount of time which is assigned to each task T_i in each period to execute the optional parts. When an optional part has exhausted this time, the optional part is aborted and later parts of the same job are not started. The reservation time is calculated on the basis of worst-case execution times for mandatory parts and the expected execution times of both the optional and mandatory parts for optional parts. Therefore the execution time of the mandatory parts M_{ij} and the optional parts O_{ijk} are represented as non-negative random variables X_{ij} and Y_{ijk}, $(k = 1, ..., c_i)$. In the context of this work, for each task T_i X_{ij} and Y_{ijk} are assumed to be identically distributed.

To summarize, with QAS a task is described by a tuple:

$$T_i = (X_i, Y_i, c_i, w_i, q_i, t), \quad i \in \mathbb{N} \tag{1}$$

where:

X_i execution time of the mandatory part;

Y_i execution time of an optional part;

c_i number of optional parts;

w_i worst-case execution time of the mandatory part, i.e., $\mathbf{P}(X_i \leq w_i) = 1$;

q_i quality parameter, percentage of successful optional parts, $0 \leq q_i \leq 1$;

t length of period.

These values must be known prior to the admission of a task. To accept a new task set, the admission control must ensure that:

1. all mandatory parts M_i meet their deadlines. This means that the worst case execution times w_i of all tasks must fit into the period:

$$\sum_{i=1}^{n} \frac{w_i}{t} \quad \leq \quad 1 \tag{2}$$

2. for each task T_i a reservation time r_i can be found so that the quality parameter q_i will be reached:

$$\exists r_1, ..., r_n \forall i = 1, ..., n : r_i = \min(r | \mathbf{E}A_i \geq q_i c_i) \tag{3}$$

$$\mathbf{E}A_i = \sum_{k=1}^{c_i} \mathbf{P}(A_i \geq k)$$

where A_i is a random variable denoting the number of optional parts of task T_i which can be executed during a period of T_i. The calculation of $\mathbf{P}(A_i \geq k)$ depends on r_i.

Note that in this calculation only the random variable X of the execution time of the mandatory part M has to be considered, not the worst-case execution time w. Thus, optional parts can be admitted even if the mandatory parts filled the whole period according to their worst-case execution times.

For a more detailed description of the admission tests and the Quality-Assuring Scheduling in general refer to [4].

3. Quality-Assuring Scheduling of Disk Requests

In this section we describe the application of Quality-Assuring Scheduling to disks. We combine a mechanism to enforce the service guarantees (i.e. the reservation times) with a disk-request scheduler which optimizes the disk utilization.

3.1. Adapting QAS to Disk Storage

Real-time data streams (e.g. video or audio streams) are mostly read or written with a nearly constant average bandwidth, for example 4 MBit/s for an MPEG-2 video stream. Given a fixed block size, this bandwidth requirement can be translated into a fixed number of disk requests that are executed periodically. The period length must obey the ability of the system to buffer blocks. Apart from that it can be chosen arbitrarily.

Identifying mandatory and optional parts within a single stream of disk requests leads to very awkward interferences. Instead, we chose to distinguish mandatory and optional data streams. However, this approach might require changes to some common applications. Figure 1 illustrates

this approach for an MPEG video application. The original stream is split into three streams, which store the different frame types of the MPEG video. To decode the video, frames are read from all three streams and reordered such that the decoder reads the frames in the expected order.

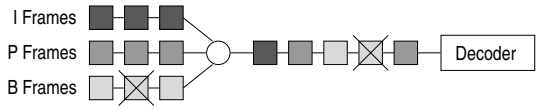


Figure 1. Splitting MPEG streams.

Applications specify the quality parameter q_i for the whole stream. For our MPEG example, an application may request that all blocks of the file containing the I frames must be read to ensure a minimum quality of the video display. Hence, this stream is requested as a mandatory stream. For the other two streams the MPEG application is modified such that it tolerates the occasional loss of disk blocks ($q_i < 1$), thus requesting the streams as optional streams. If a block of such a stream could not be retrieved on time, it is discarded as shown in Figure 1 for one block of the B-Frame stream.

In contrast to the resource CPU, the variation of the execution times is caused by the disk, the resource itself, not by the application. Hence, the generic task description of QAS (Definition 1) is mapped to a task description T_i and a description of the system D, which includes the characterization of the disk as well as the period length:

$$\begin{aligned} T_i &= (c_i, q_i), \qquad i \in \mathbb{N} \\ D &= (X, w, t) \end{aligned} \tag{4}$$

where:

c_i number of disk requests in each period;

q_i requested quality for the stream; $q = 1$ denotes a mandatory stream, $q < 1$ an optional stream;

X execution time of a disk request;

w worst-case execution time of a disk request;

t period length.

With the knowledge of w and X we can perform the admission test as described in Section 2 and calculate the reservation times for the optional streams.

3.1.1. Worst-Case Execution Time w As a basis for the worst-case assumptions we use a disk model to identify the longest possible time any disk request may require for a particular disk. A model which can accurately predict the behavior of a disk would be too complex, if it could be found at all. Additionally, such a model would require an in-depth knowledge of the disk-drive design and firmware, which in general is not available. Therefore, our approach is to use a slightly simplified model and to run a set of microbenchmarks to extract the necessary parameters for that model.

Our model is:

$$\begin{aligned} w &= t_{seek} + n * t_{rot} + m * t_{sector} \\ &\quad + t_{ovh} + v * t_{skew} \end{aligned} \tag{5}$$

where:

t_{seek} the seek time; the worst case is the time for moving the disk heads from the innermost to the outermost cylinder of the disk.

$n * t_{rot}$ the maximum rotational delay; in rare cases the disk head needs several (n) rotations to settle on the target track.

$m * t_{sector}$ the time to access the data; this depends on the number m of sectors which are accessed and the time t_{sector} to access a single sector. With modern disk drives, t_{sector} varies for different zones of the disk. The worst case for such disks is in the innermost, slowest zone of the disk.

t_{ovh} the overhead caused by the request processing in the disk controller and the data transfer between the disk and the host system. It is assumed to be a constant value.

$v * t_{skew}$ the time to switch to the next cylinder respectively the next disk head. The maximum number v of cylinder or head switches that may occur depends on the maximum request size and the minimum size of a single track.

These parameters must be acquired once for each disk in the system. For those measurements, the microbenchmarks are adjusted to deliver the worst-case values for each parameter separately. Section 4.1 shows how this is done.

Whereas it is possible to adjust microbenchmarks to trigger worst-case situations for a particular disk parameter, the measurement of the worst-case execution time of a whole disk request is not applicable. It would require a workload which causes a worst-case behavior of the disk.

3.1.2. Service-Time Distribution X In contrast to the worst-case execution time, we can use direct measurements to obtain the distribution of the request service times. Like with the worst-case parameters of a disk, the measurement of the service time distribution has to be performed once for each disk in the system. The workload used for the measurements must represent the characteristics (request locations and sizes, read or write requests) of the expected load on the system.

3.2. Enforcing Service Guarantees

A straightforward approach to schedule requests with different service guarantees is to schedule them separately:

for instance by executing all real-time requests at the beginning of each period to enforce the real-time guarantees and by assigning the remaining time in the period to best-effort requests. However, with such an approach the scope of optimizing the request scheduling is limited to each request class. This leads to a poor resource utilization, especially in the case of a large number of different classes. To exploit all optimization potential, a request scheduler must be able to choose from as many outstanding requests as possible.

In the optimal case the scheduler considers the requests of all mandatory and optional streams as well as best-effort requests. However, the scheduler must also ensure the service guarantees: execution of all mandatory streams and optional streams according to the assigned reservation time.

We ensure these guarantees in the following way: At each point in time when choosing the next request for execution, we construct a subset of the outstanding requests such, that no service guarantee will be violated regardless of which request of that subset is executed. The subset contains all *active* requests which can be considered by the scheduler. We call it *Dynamic Active Subset (DAS)*.

Figure 2 shows the algorithm for constructing the DAS. `t_left` denotes the remaining time until the end of the current period, `t_w` the worst-case execution time w. First, all outstanding mandatory stream requests for the current period are added to the DAS, and the remaining time is decreased by the worst-case execution times. If there is time left for another worst-case disk access in the period, all requests of optional streams which have not yet exhausted their reservation time are added to the DAS and `t_left` is decreased by their remaining reservation time `r_left`. `r_left` is set to the reservation time r_i at the beginning of each period and decreased by the actual execution time of each request of the stream which is executed. Finally, if there is still time left in the period, best-effort requests are added to the DAS. Note that once an optional stream is accepted for the DAS, all available requests of that stream for the current period are added to the DAS; likewise for best-effort requests, once they are accepted all available best-effort requests are added to the DAS.

Using the worst-case execution time to decide if there is enough time in the period to add more requests ensures that all service guarantees are met. Consider the following example: A mandatory stream has three outstanding requests for the current period, the remaining time in the period is 100 ms and the worst-case execution time of a request is 30 ms. Thus, `t_left` is 10 ms after adding the mandatory stream to the DAS. In this situation, no other requests can be added to the DAS, because otherwise if another request is selected for execution there might not be enough time left to execute the mandatory requests afterwards. This means Condition 2 of the admission control would be violated. Likewise for optional streams, we must ensure that all streams get their reservation time assigned before we can add best-effort requests.

```
das.init()
period = get_current_period()
t_left = period.time_left()
t_w    = disk_model.t_w

// 1. add mandatory streams
for each m_stream in period
  das.add(m_stream)
  t_left -= t_w * m_stream.req_left

// 2. add optional streams
if (t_left >= t_w)
  for each o_stream in period
    if (o_stream.r_left > 0)
      das.add(o_stream)
      t_left -= o_stream.r_left

// 3. add best-effort requests
if (time_left >= t_w)
  das.add(best-effort)
```

Figure 2. Selecting requests for the Dynamic Active Subset (DAS)

The above example also shows the benefit of this approach. In most cases, the actual execution time of a request is much smaller than the worst-case time, especially for modern disks. For instance, assume an execution time of 8 ms for one of the mandatory request which was selected for execution. Thus, there are 92 ms left in the period, which means that we can also add other requests to the DAS in addition to the two remaining mandatory requests for the next scheduling decision. This gives the request scheduler more options to optimize the disk utilization.

The algorithm to create the DAS can be expanded further to meet additional requirements, for example how optional-stream requests are treated in case the reservation time for their stream is already exhausted. The algorithm shown in Figure 2 skips such requests. However, they could also be added to the DAS as best-effort requests or they could even be given precedence over best-effort requests to improve the quality of optional streams before best-effort requests are considered.

Creating the DAS ensures that no service guarantees are violated regardless of which request of the subset is executed. This enables a request scheduler to apply an arbitrary policy to select requests out of the DAS for execution. We will now show how this can be used to optimize the disk utilization.

3.3. SATF for DAS

The DAS mechanism is designed such that, within a DAS, all scheduling algorithms can be used, while still en-

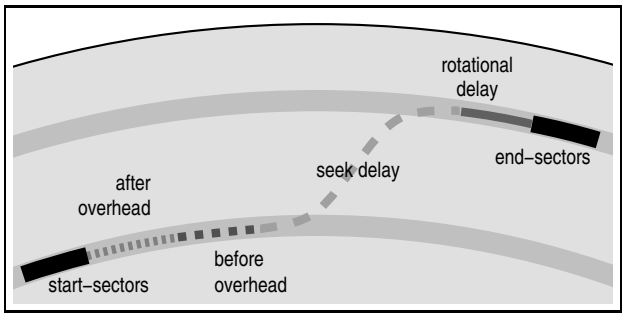


Figure 3. Progression of two subsequent disk requests.

suring all deadlines.

However, due to the highly dynamic nature of DAS, it is not possible to use the scheduling mechanisms implemented in the disk controller's hardware, as the DAS sometimes shrinks, and thus requests would have to be revoked from the disk. Furthermore, the accounting of service times for single requests and the implementation of arbitrary scheduling policies are both infeasible when using the hardware-internal scheduling.

Thus, we integrated an SATF scheduler (*shortest access time first* as described in [8]) into the disk driver, which minimizes both the seek delay (as SSTF (*shortest seek time first*) does) and the rotational delay between subsequent requests.

To implement such an algorithm outside the disk, the scheduler has to use physical disk-sector addresses rather than logical ones. This means a substantial increase in the amount of information needed to conduct scheduling in contrast to conventional algorithms using logical addresses.

3.3.1. Modeling With SSTF the overall time for a request sequence is minimized by choosing the subsequent request based on the time required to perform the seek to this request's cylinder. As the mapping from cylinder distance to time can be assumed to be monotonic, one can choose the subsequent request to be scheduled based on the cylinder distance. Furthermore, as the logical distance between two logical block addresses maps nearly monotonic to the (physical) cylinder distance, often the logical distance is used and no conversion to physical addresses is required.

This approach is not feasible when implementing SATF as information about the rotational position of sectors and the actual position of the disk head is needed.

Thus, to choose the subsequent request from the ready queue we need a model which predicts the time between the current request and the subsequent request. This is the time we aim at minimizing with SATF.

Based on the disk behavior as shown in Figure 3 and experiences from previous work such as [19, 6, 7, 1, 8, 15] we used the following model:

$$t_{between} = t_{seek}(\delta_{cyl}) + t_{ovh}(rw_{this}, rw_{next}) + t_{rot}\left(\delta_{angle}, t_{ovh}(rw_{this}, rw_{next}), t_{seek}(\delta_{cyl})\right) \quad (6)$$

where:

$t_{seek}(\delta_{cyl})$ is the time necessary for the seek between the cylinder of the current request and the subsequent request. It depends on the cylinder distance δ_{cyl}. As single requests may span several tracks the computation must be based on the physical position of the last sector of the current request and the first sector of the subsequent request.

t_{ovh} denotes the overhead time occurring between the current request and the subsequent request. It consists of two parts, the overhead occurring after the current request and the overhead before the subsequent request. Both are not fixed but depend on the types of requests issued. We noticed different overhead times for read and write requests.

The different times of the head settling for read and write accesses to the platters after seeks might be responsible for this difference [14]. Another difference between read and write requests is the amount of data to be transferred before and after the requests. For read requests, no data has to be transferred before the command can be started but the last sectors read by the disk must be transferred to the host after completion of the command, whereas at least data for one sector has to be transferred to the disk before it can start a write command. On completion of a write request no data, but only a completion message needs to be transferred to the host controller.

A correlation between request size and overhead could not be detected for SCSI disks. The explanation for this is that due to the parallel execution of requests and data transfers between host controller and disk these overheads are nearly fixed as only the first sectors for write requests and the last ones for read requests need to be transferred.

Similar investigations for ATA disks using a Linux device driver showed a clear dependency between the request sizes and occurring overhead times. Thus, this model will have to be changed if applied to ATA disks.

$t_{rot}\left(\delta_{angle}, t_{ovh}, t_{seek}(\delta_{cyl})\right)$ models the rotational delay the disk head has to wait for the destination sector to arrive after head settling is finished and the overhead time has elapsed. The value of this parameter is between zero and one full rotation of the disk (in case the destination sector was just missed) and depends on the angle difference δ_{angle} between the current request and the subsequent request, the overhead time t_{ovh}, and the seek time $t_{seek}(\delta_{cyl})$. As for the seek time, also for the rotational delay, the last sector of the current request

and the first sector of the subsequent request are the starting point for the computation.

3.3.2. Necessary Information To be able to implement SATF scheduling based on the model described above we need static geometry data, which we assume to be fixed (remapping of defect sectors at runtime is ignored in our actual implementation) and dynamic information, which has to be updated at runtime.

The following data belongs to the *static part*:

- The complete mapping from logical to physical addresses is used to determine the cylinder addresses and the angle offsets of specific sectors.
- The rotational speed of the disk is used for the computation of angle positions.
- The seek curve which maps from cylinder distance to time is used to predict the time for specific seeks.
- We distinguish overhead times for four cases: read-read, read-write, write-read and write-write request pairs. These four times have to be acquired for the desired *Tagged Command Queueing (TCQ)* queue sizes. Reasonable are sizes of one and two (see Section 3.3.3).

For the *dynamic part* of the disk model we need to maintain the internal state of the disk. For SATF the relevant information is the current position of the disk head and the type of the current request, so that we can choose the right overhead time for the prediction of the service time for the subsequent request.

In previous work, different approaches to keep track of the head position were discussed. The most complex ones compute the rotational position from the actual time, the rotation time of the disk and regular calibration requests [7, 9]. However, this involves additional requests sent to the disk as well as an accurate time source.

In contrast, in our implementation we use the physical position of the current request and the model described in Section 3.3.1 to compute the actual position of the disk head. This works always except in cases when the disk has idled some time. In this situation the rotational position is unknown to the scheduler. On the other hand, in such situations the request queue is empty and therefore the scheduler has no assortment of requests to choose from anyway. So the simple approach based on the previous request seems appropriate.

3.3.3. Hiding the Overhead Times In the situation of not using TCQ we spend a substantial amount of time in communication with the disk without media access. This is necessary as the disk needs to know what to do and the data to be written, while the host needs the data read by the disk.

In the following we describe the usage of TCQ to minimize these times as proposed in [9].

With modern SCSI and ATA disks it is possible to send more than one request to the disk concurrently. The disk normally executes one request at a time, so the internal request queue can be kept filled by the host. For current SCSI disks we determined a maximal queue size of 32. This size should be enough to hide the overhead times introduced by communication between host controller and disk.

In fact, a request queue size of two is sufficient. One request is transferred to or from the disk while the other is being executed. When using larger request-queue sizes there are times when the disk has two or more ready requests in its queue and therefore the disk could use its internal scheduler to reorder the execution sequence. Thus, using a queue size of two has the advantage that we can keep full control over the deadlines.

With SCSI disks there is another way, besides limiting the hardware queue size, to prevent the disk from reordering requests, the use of *ordered queue tags*. Each outstanding request queued in the disk can be tagged with either a *head of queue tag*, an *ordered queue tag*, or a *simple queue tag* [17]. The *ordered queue tags* act as barriers in the sequence. If each request sent to the disk is tagged with this tag the disk is not allowed to reorder. *Simple queue tags* raise no restrictions regarding the reordering of requests. If requests tagged with the *head of queue tag* are sent to the disk they are inserted as the request to be executed next.

However, using these tags earns no advantages beyond what can be achieved with the limitation of the queue size, thus they are not investigated further in this paper.

3.4. Starvation and Response Times

We have designed DAS with the intent that any scheduling algorithm can be used in combination with it and real-time guarantees are still kept. In our first implementation we concentrated on the overall system performance and therefore chose SATF. However, a well known disadvantage of SATF is that it does not prevent starvation. Since DAS guarantees that deadlines for real-time requests are met, starvation can only occur for best-effort requests.

Several generic ways of adding starvation control to scheduling algorithms can be found in the literature. Such methods, namely *aging* and *grouping*, are described in [15]. Similarly to the mentioned methods we slightly modified the SATF implementation to have control over maximum latency and therefore to prevent starvation completely.

The modification consists of a threshold governing how long single requests are allowed to stay in the driver's queue. If they stay longer, the modified SATF prefers them over requests which could be reached faster according to the unmodified SATF.

This algorithm is known under the name SATFUF (*shortest access time first with urgent forcing*) and is de-

	IBM Ultrastar 36Z15 IC35L018UWPR15	Seagate Barracuda 36ES2 ST318438LW
Size	18.4 GB	18.4 GB
RPM	15 000	7 200
t_{seek}	7.18 ms	11.36 ms
t_{rot}	4.02 ms	8.37 ms
n	5	2
t_{sector}	0.01 ms	0.01 ms
m	128	128
t_{ovh}	0.07 ms	0.50 ms
t_{skew}	1.06 ms	1.23 ms
v	1	1
w	**29.69 ms**	**31.11 ms**

Table 1. Disk and worst-case parameters

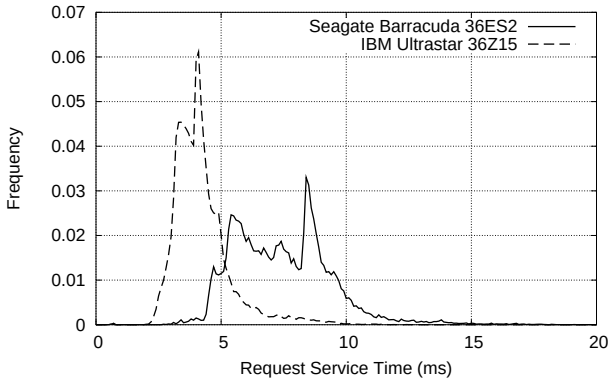


Figure 4. Request service time distribution.

scribed in [8]. There are more sophisticated ways to implement starvation control, but this simple approach is adequate to demonstrate the feasibility of software control for starvation in our system.

Starting from the performance-oriented approach we implemented, it seems easy to implement arbitrary policies. For instance, one might generally prefer best-effort requests over real-time requests in the scheduler so as to increase the responsiveness of the interactive best-effort system.

4. Evaluation

To evaluate the presented scheduling framework, we integrated the scheduler into the SCSI driver of DROPS [5], which we ported from Linux. For the measurements we used a 1 GHz Pentium 3 PC with 256 MB of RAM and two different disks, an IBM Ultrastar 36Z15 and a Seagate Barracuda 36ES2 disk drive, which were connected to a Tekram DC-390U3W Ultra 160 SCSI adapter.

4.1. Disk Parameter Extraction

To gain the necessary information about the physical layout and characteristics of the disks, we implemented a set of microbenchmarks. This work was inspired by [1] and is described in more detail in [13, 12].

The extraction mechanisms work empirically and are able to gain information from ATA and SCSI disks. To extract this information we instrumented the disk drivers in the Linux kernel. A similar approach is described in [3].

Using the methods described in the literature mentioned above, we gained most of the information for the worst-case prediction, which is outlined in Table 1, the seek curve, and the physical disk layout. The distributions of the request service times for the two disks are shown in Figure 4. They were measured for a random workload of 64 KByte read requests.

Additionally, we used two techniques which we will describe below. The first is the determination of the overhead times t_{ovh} for Model 6 and the second is the enquiry of the number of additional rotations n for Model 5.

Overhead Times The overhead time used in Model 6 consists of two parts, the overhead before and the overhead after a request, as described in Section 3.3.1.

To determine these two times for read and write requests we conducted the following experiment: We issued a random workload containing both read and write requests to the disk. We initialized the SATF model with default values for the overhead-time parameters and scheduled the workload with this SATF. We measured the resulting bandwidth while using SATF and varying each of the four time parameters independently. For each of the parameters we chose the value resulting in the highest bandwidth.

Further experiments with SCSI disks showed that the overhead times do not depend on the request size (see Section 3.3.1). We repeated the above experiment using different block sizes and obtained similar results.

Parameter n In rare cases the disk head needs some additional rotations to settle on the destination track. To obtain the maximal number of additional rotations we provoked worst case situations by alternately issuing requests to the innermost and outermost region of the disk.

In this experiment we measured the times needed to complete the requests. With the help of Equation 5 we computed the n for the worst case. To obtain realistic numbers we issued a large number of requests (at least 100 000).

4.2. Creation of the DAS

To understand how the DAS varies over time, we used a setup with three different streams, one mandatory stream with 5 requests per period, one optional stream with 45 requests per period and a best-effort stream with a constant load of 100 requests. We ran this setup and measured the

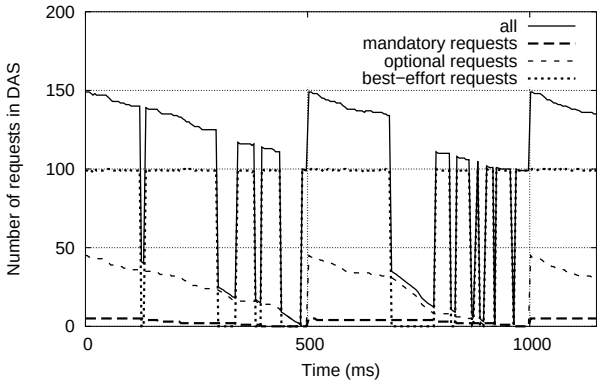


Figure 5. Number of requests in the DAS (period 500 ms, random load, IBM Ultrastar).

number of requests of the mandatory, optional and best-effort streams which were added to the DAS each time it was created.

Figure 5 shows the result of this experiment. At the beginning of the period the requests of all streams are added to the DAS, because the mandatory and optional streams do not completely fill up the period. The total number of requests in the DAS decreases over time as more of the mandatory and optional requests are executed (remember that the number of outstanding best-effort requests is fixed). At some points only the requests of the real-time streams are added to the DAS, reducing the total number of requests in the DAS. This ensures that their service guarantees can be met and happens more frequently towards the end of the period. In addition to the number of requests in the DAS we measured the time required to calculate the DAS, we measured an overhead of about 30 000 cycles (30 µs on our CPU). Even though this overhead occurs on each scheduling decision, it is still negligible compared to the overall costs of a disk request.

4.3. SATF

As described in Section 3.3, an SATF scheduler requires the mapping of logical block addresses to the rotational positions of the sectors on the disk. One such translation takes about 1 500 cycles (1.5 µs). It is done prior to the request scheduling, thus it adds no significant overhead to a scheduling decision.

4.3.1. Accuracy of the Model To verify the accuracy of Model 6 we issued a large number of small requests to the disk and measured the difference between the predicted time to the subsequent request according to the model and the real execution time.

The results of this experiment are shown in Figure 6. When using no TCQ a small proportion of the requests is

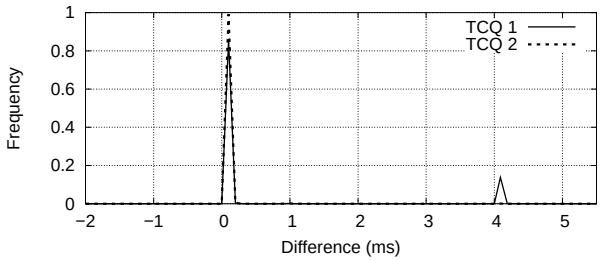


Figure 6. Accuracy of request service time prediction (4 ms rotation time)

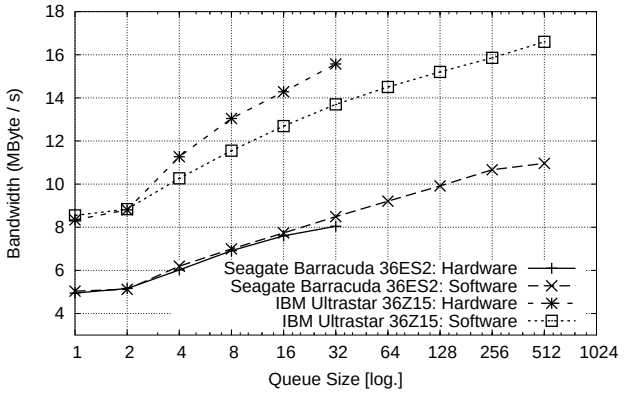


Figure 7. Scheduler performance

completed one rotation of the disk later (about 4 ms for the IBM disk), while the majority is executed as predicted.

A dramatic increase in accuracy is achieved when using TCQ with a size of two. Nearly all requests are executed as predicted. So, using this technique of hiding the command overhead not only increases the disk utilization but also the accuracy of the prediction.

4.3.2. Performance Evaluation For performance evaluation we compared our SATF implementation with the disk-internal scheduler using TCQ. With the investigated disks we experienced a queue size limit of 32 when using TCQ. In software much larger queues are feasible.

Figure 7 shows the performance results for a random workload for both disks. On the very fast rotating IBM disk SATF loses about 12 % compared to the disk internal scheduler at a queue size of 32, but still outperforms the disk with larger queue sizes. With the relatively slowly rotating Seagate disk SATF is about 6 % faster with a queue size of 32.

4.3.3. Latency To gain an understanding of the magnitude of request-execution latencies that can be caused, a very simplistic experiment was performed: Two best-effort request streams are started concurrently, both consisting of read and write requests and both using random block addresses. The first stream addresses the first logical quarter

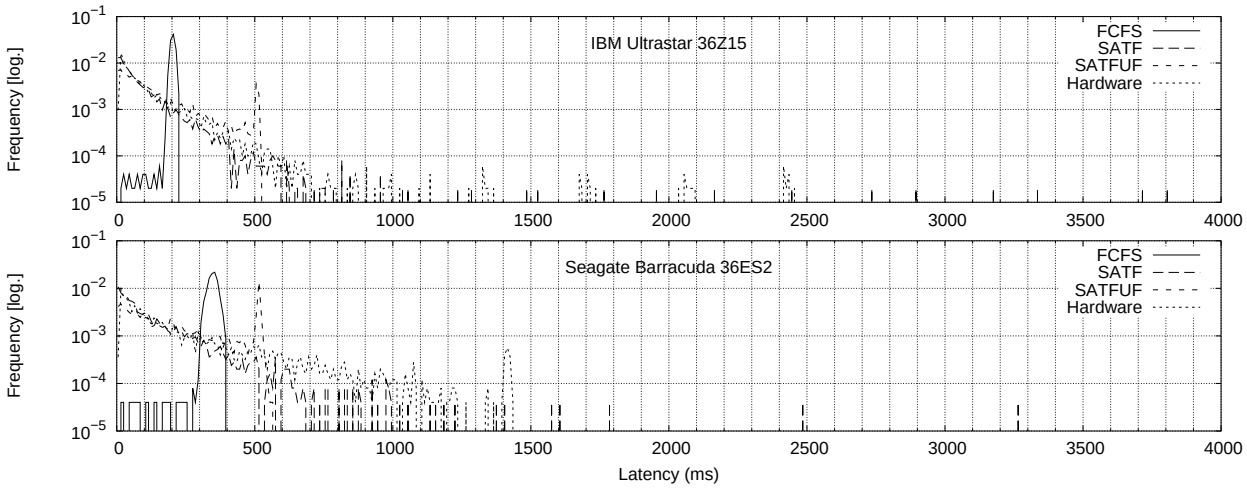


Figure 8. Latency for different scheduling algorithms.

of the disk and uses 100 times more bandwidth than the second, while the second stream addresses the very end of the disk.

We expect that the second stream will starve or that requests from that stream will be delayed for very long times, depending on the scheduling algorithm used.

The results of this experiment are shown in Figure 8 for both disks. SATF has the largest latency of about 10 s (maximum not shown) in our experiments. This is to be expected as there is no starvation control at all.

Using the disk-internal scheduling we still obtain very long times of about 2 400 ms for the IBM disk and about 1 400 ms for the Seagate disk.

Using SATFUF as described in Section 3.4 we can control the maximal starvation time easily by modifying the threshold. We can very accurately trade overall system performance for responsiveness in the best-effort part of the system. For the experiments shown in the graph we used a threshold of 500 ms.

Using FCFS (*first come first served*) results in very compact distributed response times for each of the disks. While known to be the best choice for responsiveness, it normally results in very bad overall system performance.

The results show that some kind of starvation control is implemented in the disks. However, the latency experienced strongly suggests some control at software level.

4.4. Overall System Behavior

4.4.1. Benefit of Statistical Guarantees Figure 9 shows the bandwidth which can be assigned by the admission control to one optional stream depending on the requested quality compared to the bandwidth which can be achieved with a best-effort load (64 KByte requests, 500 ms period length,

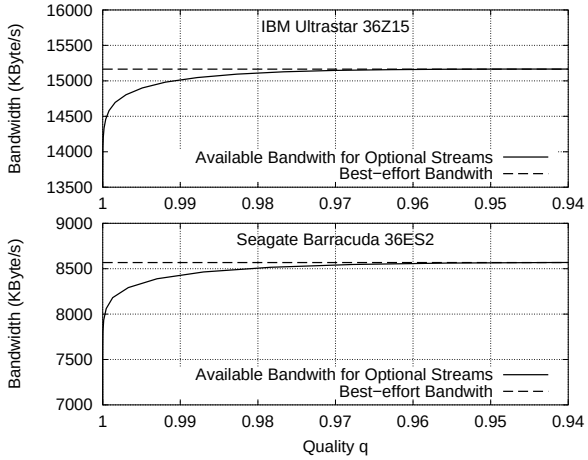


Figure 9. Max. bandwidth which can be assigned to an optional stream.

random load). The maximum bandwidth of the optional stream was determined on the basis of Condition 3 of the admission control using the service-time distributions shown in Figure 4. The graph shows the bandwidth which can be actually used by an application, that is $q_i * \frac{b * c_i}{t}$ where b denotes the block size of the requests.

This measurement demonstrates the large benefit of using statistical guarantees with disk drives. With a worst-case execution time of about 30 ms for both disks (see Table 1) only a bandwidth of 2 MByte/s can be assigned to real-time streams, whereas the penalty of using statistical guarantees is much lower. With a requested quality of 99.9 %, 14.6 MByte/s or 96 % of the best-effort bandwidth can be used for the IBM disk, for the Seagate disk 8.1 MByte/s or

94 %. With a quality of 95.5 % for the IBM disk and 94.7 % for the Seagate disk the whole bandwidth can be assigned to real-time streams.

4.4.2. Service Guarantees and Disk Utilization With the next measurement we want to evaluate the effects of enforcing service guarantees on the disk utilization. As we have already shown in Section 4.2, the DAS does not always contain all outstanding requests. To measure how this affects the disk bandwidth we used a setup consisting of one real-time stream and a best-effort load with a constant number of outstanding requests. We measured the bandwidth we achieved for both streams and additionally the portion of scheduling decisions where only the real-time requests were added to the DAS. For the first part of this measurement we used a mandatory stream. Figures 10(a) and 10(b) show the bandwidth we achieved depending on the number of requests we used for the mandatory stream. As it can be seen, enforcing the guarantees for a mandatory stream has no effect on the overall bandwidth, although in about 5 % respectively 12 % of the scheduling decisions only the requests of the mandatory stream were considered by the scheduler. Because the mandatory stream had to be admitted using worst-case execution times it could only occupy a small portion of the available bandwidth (max. 2 MByte/s for both disks). Therefore we replaced the mandatory stream with an optional stream ($q = 0.95$) for the next measurement. Again, the results shown in Figures 10(c) and 10(d) show that there is no penalty for enforcing the statistical guarantees. Instead, the overall bandwidth even slightly increases.

These results are partially caused by the random workload we use in our measurements. With such a workload the bandwidth which can be achieved by the scheduler depends mainly on the number of requests it can choose from. For the mandatory load the small number of scheduling decisions, where only the requests of the mandatory streams were added, do not have an effect on the overall bandwidth. For the optional stream this explains the slight increase in the bandwidth, especially if the period is shared equally between the optional stream and the best effort load. In this case the requests of both the optional stream and best-effort load can be added to the DAS most of the time, whereas for a larger number of optional requests only the requests of the optional stream are added to the DAS, decreasing its average size.

To verify that our scheduler also works for other workloads, we ran the same measurement for a sequential rather than a random workload. Figures 10(e) and 10(f) show that also for this load there is no penalty for enforcing service guarantees.

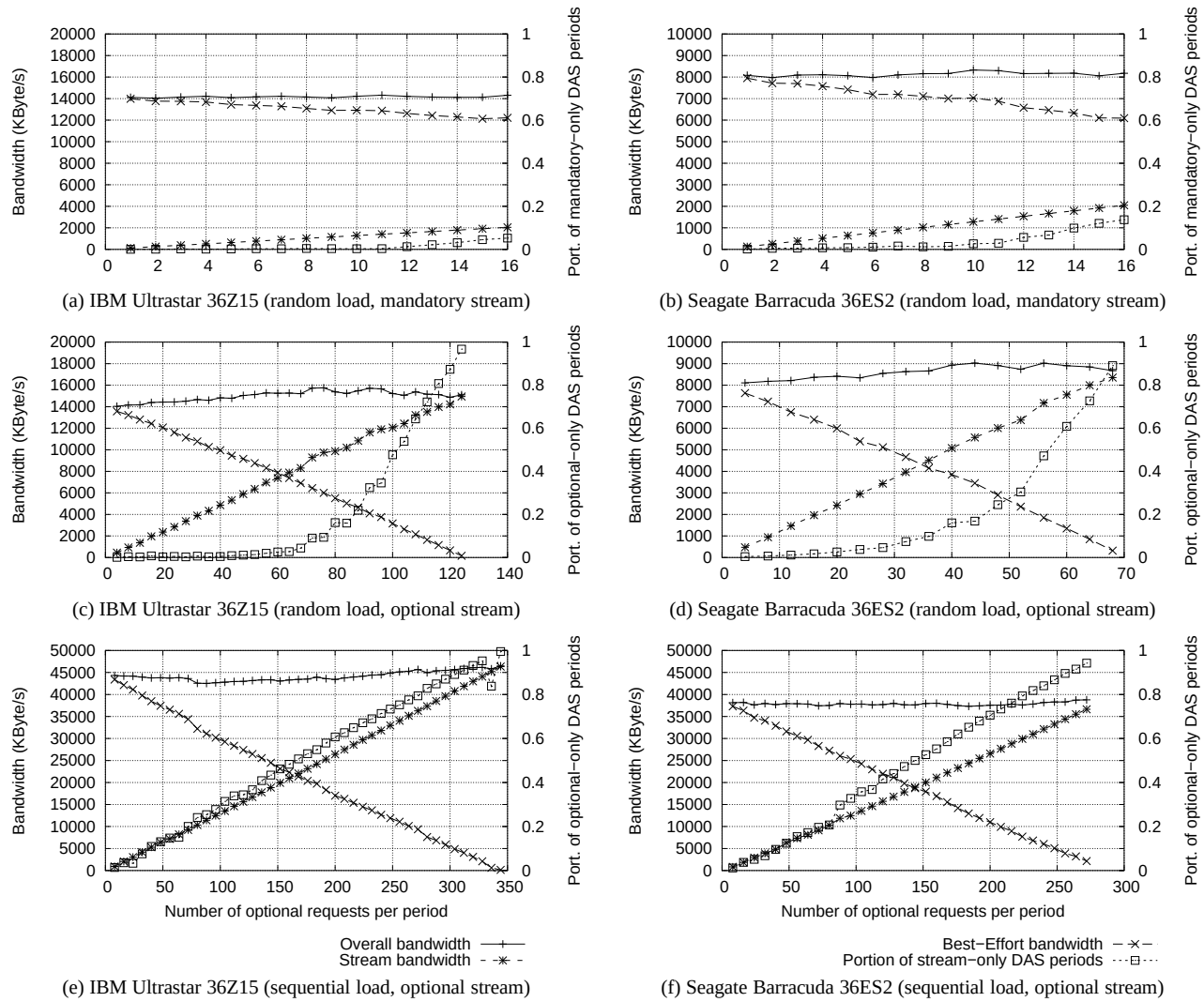
4.4.3. A Complex Example To summarize our evaluation we ran a complex setup. Table 2 shows this setup, it contains five optional streams with different quality requirements and an additional best-effort load. As you can see our scheduling framework fulfills the requested qualities and assigns the remaining bandwidth to the best-effort load. The bandwidth which is assigned to the best-effort load depends on the ability of the particular disk. With the IBM disk we achieve a total bandwidth of 16.3 MByte/s and with the Seagate disk we achieve 9.3 MByte/s.

5. Related Work

Algorithms using a similar approach to the calculation of the DAS are slack-time-based algorithms like the Just-in-Time scheduler [10] or the ΔL scheduler [2]. In general, the slack time of a request is the difference between the latest time by which the request must be executed and the earliest time the request can be executed. Within this interval, a scheduler can choose any time to execute the request. The ΔL scheduler tries to improve the responsiveness of best-effort requests by giving them precedence over real-time requests whenever sufficient slack time is available. The physical positions of requests are not considered at all by the ΔL scheduler, real-time requests are scheduled in EDF order and best-effort requests in FCFS order. The Just-in-Time scheduler uses the slack time to schedule requests in SCAN order to improve the disk throughput.

The Cello Disk-Scheduling Framework [16] aims to meet the service requirements of different types of applications, for example periodic real-time, interactive best-effort, or high-throughput best-effort applications. It uses a two-level scheduling scheme: a class-independent scheduler which distributes the available bandwidth among the different request classes and a class-specific scheduler for each request class which schedules the requests of that class according to the application requirements. The class-independent scheduler maintains a scheduled queue which contains the requests which have been already selected for execution. The state of this queue is exported to the class-specific schedulers in terms of slack time for each request. These schedulers can use this information to implement for instance a slack-stealing algorithm similar to [2] for interactive best-effort applications or a Just-in-Time EDF scheduler for real-time applications. The evaluation of Cello was done using several simulations. The paper identifies two problems with this approach, one being that the order in which the class-specific schedulers are invoked impacts the feasibility of the request schedule and the disk utilization. Our approach does not suffer from this problem, we separate the request scheduling from the enforcement of service guarantees.

Various systems use statistical guarantees for disk requests in the context of multimedia or continuous-media servers [18, 11]. [18] describes an admission-control and scheduling algorithm for continuous-media streams similar



(a) IBM Ultrastar 36Z15 (random load, mandatory stream)

(b) Seagate Barracuda 36ES2 (random load, mandatory stream)

(c) IBM Ultrastar 36Z15 (random load, optional stream)

(d) Seagate Barracuda 36ES2 (random load, optional stream)

(e) IBM Ultrastar 36Z15 (sequential load, optional stream)

(f) Seagate Barracuda 36ES2 (sequential load, optional stream)

Figure 10. Influence of real-time streams on disk utilization.

to ours; disk request are scheduled in rounds, new streams are only accepted if at least a client-specified percentage of frames of the stream can be delivered on time. Our system extends this by also considering hard real-time and best-effort applications, which is a crucial requirement for today's systems having to cope with varying application demands.

6. Conclusion and Outlook

In this paper we presented admission-control and scheduling mechanisms for disk requests supporting hard real-time applications, statistical real-time applications, and best-effort applications. Our scheduling algorithm separates the enforcement of QoS guarantees (i.e. the calculation of the DAS) from the optimization of the disk utilization. With this separation various disk schedulers can be implemented without violating guarantees. We demonstrated this with the modifications of our original SATF scheduler to provide better response times for best-effort requests. The evaluation of our implementation pointed out that enforcing service guarantees does not affect the ability of a request scheduler to optimize the disk utilization and the benefits of using statistical service guarantees.

Future work will include the consideration of block-placement policies of file systems on the admission control. In our experiments we used a synthetic workload which was uniformly distributed over the whole disk. Incorporating knowledge of the placement policy can further improve the disk utilization. The performance results we presented in this paper represent a slightly pessimistic lower bound. One of the problems to be solved is the creation of a request-service-time distribution representing file-system behavior.

				IBM Ultrastar 36Z15			Seagate Barracuda 36ES2		
Stream (i)	Requests / Period (c_i)	Band-width KByte/s	Requested Quality (q_i) %	Reservation Time (r_i) ms	Achieved Quality %	Achieved Bandwidth KByte/s	Reservation Time (r_i) ms	Achieved Quality %	Achieved Bandwidth KByte/s
1	5	640	99.00	20.90	99.73	638.54	36.90	98.83	632.81
2	20	2560	95.00	77.50	97.98	2509.12	139.00	96.88	2481.30
3	10	1280	90.00	35.60	94.44	1209.36	63.50	93.58	1198.39
4	5	640	85.00	15.80	90.04	576.44	28.00	90.30	578.13
5	10	1280	60.00	23.40	67.05	858.65	41.00	67.44	632.81
Best-effort bandwidth						10473.13			3528.73
Total bandwidth						16265.28			9282.93

Table 2. Complex Example

One approach could be to replay request traces, but traces cannot be applied to different disks easily.

Future work will also include the extension of our scheduling algorithm to support real-time applications with arbitrary deadline requirements for disk requests. The overall idea is to modify the calculation of the DAS to use the deadline of the closest request instead of the period end. In general, by adapting the calculation of the DAS, different policies can be implemented, for example the distribution of the remaining time in a period which is not assigned to real-time streams among all applications.

Several extensions to the model used for the SATF time prediction are possible, for instance modeling of the disk-cache behavior, adaption to the "access on arrival" feature of modern SCSI disks, adaption to overhead times depending on disk-request sizes for ATA disks, and considering the varying interrupt latencies in existing real-time systems for the prediction of the actual disk head angle. These should be investigated in further studies.

References

[1] M. Aboutabl, A. K. Agrawala, and J.-D. Decotignie. Temporally Determinate Disk Access: An Experimental Approach. In *Measurement and Modeling of Computer Systems*, 1998.

[2] P. Bosch and S. J. Mullender. Real-Time Disk Scheduling in a Mixed-Media File System. In *Sixth IEEE Real Time Technology and Applications Symposium (RTAS)*, 2000.

[3] Z. Dimitrijevi, R. Rangaswami, E. Chang, D. Watson, and A. Acharya. Diskbench. Technical report, 2001.

[4] C.-J. Hamann, J. Löser, L. Reuther, S. Schönberg, J. Wolter, and H. Härtig. Quality Assuring Scheduling - Deploying Stochastic Behavior to Improve Resource Utilization. In *22nd IEEE Real-Time Systems Symposium (RTSS)*, 2001.

[5] H. Härtig, R. Baumgartl, M. Borriss, C.-J. Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schönberg, and J. Wolter. DROPS: OS Support for Distributed Multimedia Applications. In *Eighth ACM SIGOPS European Workshop*, 1998.

[6] L. Huang and T. Chiueh. Implementation of a Rotation Latency Sensitive Disk Scheduler. Technical Report ECSL-TR81, State University of New York, Suny Brook, 2000.

[7] L. Huang and T.-C. Chiueh. Experiences in Building a Software-Based SATF Scheduler. Technical report, State University of New York, Suny Brook, 2002.

[8] D. M. Jacobson and J. Wilkes. Disk Scheduling Algorithms Based on Rotational Position. Technical Report HPL-CSP-91-7rev1, HP Laboratories.

[9] C. R. Lumb, J. Schindler, and G. R. Ganger. Freeblock Scheduling Outside of Disk Firmware. In *Conference on File and Storage Technologies (FAST)*, 2002.

[10] A. Molano, K. Juvva, and R. Rajkumar. Real-Time Filesystems: Guaranteeing Timing Constraints for Disk Accesses in RT-Mach. In *IEEE Real-Time Systems Symposium (RTSS)*, 1997.

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

[12] M. Pohlack. Ermittlung von Festplatten-Echtzeiteigenschaften, 2002. Undergraduate thesis (in German).

[13] M. Pohlack. Plattenscheduling für Quality-Assuring-Scheduling, 2003. Master's thesis, (in German).

[14] C. Ruemmler and J. Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer*, 27(3), 1994.

[15] M. Seltzer, P. Chen, and J. Ousterhout. Disk Scheduling Revisited. In *USENIX Winter Technical Conference*, 1990.

[16] P. Shenoy and H. M. Vin. Cello: A Disk Scheduling Framework for Next Generation Operating Systems. In *ACM SIGMETRICS Conference*, 1998.

[17] Technical Committee T10. Information Technology - Small Computer System Interface (SCSI) - 2, 1996.

[18] H. Vin, P. Goyal, A. Goyal, and A. Goyal. A Statistical Admission Control Algorithm for Multimedia Servers. In *ACM Multimedia*, 1994.

[19] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling Algorithms for Modern Disk Drives. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1994.