Cluster Computing 6, 75–86, 2003
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

Clustered Scheduling Algorithms for Mixed-Media Disk Workloads in a Multimedia Server

ELIAS BALAFOUTIS, MICHAEL PATERAKIS and PETER TRIANTAFILLOU *
Department of Computer and Electronics Engineering, Technical University of Crete, Greece

GUIDO NERJES
Bertelsmann MediaSystems, Guetersloh, Germany

PETER MUTH and GERHARD WEIKUM
Department of Computer Science, University of Saarland, Germany

Abstract. Divisible load scenarios occur in modern media server applications since most multimedia applications typically require access to continuous and discrete data. A high performance Continuous Media (CM) server greatly depends on the ability of its disk IO subsystem to serve both types of workloads efficiently. Disk scheduling algorithms for mixed media workloads, although they play a central role in this task, have been overlooked by related research efforts. These algorithms must satisfy several stringent performance goals, such as achieving low response time and ensuring fairness, for the discrete-data workload, while at the same time guaranteeing the uninterrupted delivery of continuous data, for the continuous-data workload. The focus of this paper is on disk scheduling algorithms for mixed media workloads in a multimedia information server. We propose novel algorithms, present a taxonomy of relevant algorithms, and study their performance through experimentation. Our results show that our algorithms offer drastic improvements in discrete request average response times, are fair, serve continuous requests without interruptions, and that the disk technology trends are such that the expected performance benefits can be even greater in the future.

1. Introduction

Multimedia servers for many applications (such as digital libraries, news-on-demand, teleteaching, etc.) will have to manage mixed-media workloads containing continuous data requests (e.g., for video and audio) and discrete data requests (e.g., for text or image data). The performance requirements that have to be met are very stringent and request-type specific. Each continuous data fragment must be delivered within a specified time limit, in order to avoid interruptions in the flow of data, termed hiccups or glitches. At the same time, the server must ensure good performance for discrete requests (such as low response times, fairness, etc.). Consequently, offering high performance to both workload types is a formidable challenge, which disk-scheduling algorithms must face. However, disk-scheduling algorithms for mixed workloads have not enjoyed a great deal of attention by related research efforts, which have mostly concentrated either on the scheduling of continuous workloads or of discrete workloads, but not both.

Multimedia servers, like all servers in a divisible load scenario, must be scalable in the sense that they can support a potentially very large number of simultaneous data streams between the server and different clients. In high-end applications with thousands of simultaneous streams the server is typically configured as a multi-disk system, possibly in a multi-computer cluster. To fully leverage the performance potential of these architectures, the load needs to be intelligently divided across the underlying disks and/or cluster nodes. Like in other applications of divisible load scheduling [2,16,17], the placement of the data and the scheduling of data requests are key for load balance and scalability. The architecture that we assume in this paper partitions large data objects such as videos and spreads the resulting data fragments across disks or cluster nodes in a striped (i.e., round-robin) or random manner. Fragments must be large enough so that each of them resides on a single node. This way, the data access load can be evenly distributed across all nodes while a data stream is being served, yet the load distribution is coarse-grained enough to achieve close-to-optimal throughput of each node. So the server perfectly scales up, in terms of the number of simultaneously sustainable streams, by adding more disks or nodes to the cluster. Perfect load division is possible under the assumption that the server delivers data fragments to its clients in a time-wise regular, round-based manner. Having to serve spontaneously arriving, discrete data requests additionally to and concurrently with the periodic, continuous data requests poses an extra challenge that is addressed in this paper. In general, the efficiency of each disk's or node's local processing has a major impact on the aggregated performance of the entire server. The focus of the current paper is on local scheduling algorithms, and this has great relevance for the scalability of a cluster-based multimedia server.

Continuous media servers provide service in time periods called "rounds" and within each round all outstanding re-

* Corresponding author.

quests for continuous data fragments must be served in order to avoid presentation glitches. However, since a typical workload of a modern media server (and each of its disk devices) typically consists of requests for continuous and discrete data, scheduling these different types of load efficiently corresponds to dividing the round time period appropriately to the different types of load, which constitutes, in essence, a divisible load scheduling scenario *in the time dimension*. In fact, one proposed scheduling algorithm divides a round into sub-rounds, one for each workload type. Hence, from this viewpoint the algorithms that follow study the problem of efficiently dividing the service round among the two different load types. Viewed from this perspective, the focus of this paper is to propose disk scheduling algorithms for these different types of workload, provide a taxonomy of relevant algorithms, and study their performance.

The organization of the paper is as follows. In the rest of this section we briefly review related work, describe the system model, and define the problem at hand. In section 2 we develop new scheduling algorithms, motivate their usefulness, and present a taxonomy of them. The simulation-based testbed and the performance evaluation of the proposed algorithms are presented in section 3. The results clearly show the benefits of our novel technique, coined clustered scheduling. In section 4 we develop alternative clustering techniques and study the performance improvements they introduce. Moreover we compare the performance of clustered scheduling against that of traditional "flat" scheduling algorithms. Finally, we study the performance of these algorithms on future disk drives, using disk technology improvement projections. These results testify that the related technology trends are in favor of the proposed clustered scheduling algorithms, whose comparative performance thus can be expected to be even better in the future. The paper is concluded in section 5.

1.1. Related work

1.1.1. Media servers on clusters

There is great consensus that the I/O subsystem of a computer system has become the performance bottleneck. This observation was the motivation for a large body of research in the area of storage servers, aiming to increase the available I/O bandwidth in the system. One thread of this research led to the development of disk arrays and to the development of new methods for placing data objects on disk arrays, which attempt to exploit their inherent potential for high performance and reliability, mainly through data placement techniques, such as striping [22]. Multimedia data, such as video and audio, require very large storage capacities and very large I/O bandwidths, making disk arrays the natural choice for secondary storage and consequently multi-disk/multi-processor systems suitable solutions for large video servers. Typical examples of such large distributed video servers are Microsoft's Tiger system [3] (a.k.a. MS Theater Server), Oracle's media server [14], Fellini/Cineblitz from Lucent [20], etc. The main advantage of these systems is that they manage to efficiently balance the disk storage and bandwidth requirements of the users across the system. This is achieved mainly by striping all server contents across all of its disks. Another key factor to efficiently exploit the distributed implementation of these systems is request scheduling. The schedule is distributed between the disks/processors, which use it to send the appropriate block of data to a viewer at the correct time.

1.1.2. Disk scheduling algorithms

Early scheduling algorithms focused on reducing seek times. The Shortest Seek Time First (SSTF) algorithm [6] achieved this; however, it is highly unfair and starvation-bound. SCAN scheduling was also proposed in [6]. It serves requests in an elevator-like manner (as it sweeps the disk in either one or both directions), allowing seek-time optimizations, while increasing the fairness. In [8] the authors proposed a parameterized generalization of SSTF and SCAN. The $V(R)$ algorithm operates as SSTF except that, every time it changes direction it adds a penalty, dependent on the parameter R and the seek distance. When $R = 1$ ($R = 0$) $V(R)$ reduces to SCAN (SSTF). The performance of these algorithms, as well as other variation of SCAN, such as the LOOK algorithm (which changes scanning direction when no more requests are pending in the current direction) has been studied experimentally in [33] and analytically in [5].

A significant development in modern-disk scheduling was to also target the high costs owing to rotational delays. These algorithms [12,29] attempted to minimize the sum of seek and rotational delays by favoring, for instance, the request with the Smallest Positioning Time First (SPTF). Recently, disk drives have been developed which can allow the exploitation of their detailed knowledge of the requested blocks' positions and minimize seek and rotational delays [23].

1.1.3. Disk scheduling algorithms for continuous data requests

The SCAN algorithm is inapplicable for continuous data requests since it is oblivious of deadlines. The Earliest Deadline First (EDF) algorithm [19] is a natural choice; however, it has poor performance since it does not attempt to reduce the overhead. SCAN-EDF [29] is a hybrid that serves requests in EDF order, but when several requests have the same deadline, they are served using SCAN. Most related recent research has adopted the notion of scheduling with *rounds* [1,9,10,22,31–33]. Each continuous data object (stream) is divided into blocks (also called fragments) such that the playback duration of each fragment is some constant time (typically, from one to a few seconds).

The round length represents an upper bound on the time in which the storage server must retrieve from disk the next fragments for all active continuous displays, or some displays will suffer a glitch. Within a round, it is possible to employ either a round-robin or a SCAN algorithm. The latter performs seek optimizations, resulting in better disk throughput. However, this is achieved at the expense of higher start-up latencies; the display cannot be started immediately after the retrieval of its first block but only after the end of the round. This is done to avoid glitches, since the service order differs from round to round. This limitation is not present when using round-robin scheduling, which also has lower RAM buffer requirements since it does not require the double-buffering scheme required by SCAN between successive rounds. A compromise was achieved with the Group Sweeping Scheduling (GSS) algorithm [33]. GSS groups streams and employs round-robin scheduling for the different groups and SCAN scheduling for the streams' blocks in a group. Thus, when there is only one group GSS reduces to SCAN and when each stream is in its own group GSS reduces to round-robin.

1.1.4. Disk scheduling algorithms for mixed-media workloads

To our knowledge, the works with some relevance to mixed-media disk scheduling are [11,18,21,24,25,27,30]. The work in [25] overviewed the performance goals for mixed workload scheduling, which are similar to ours, and studied only how some known algorithms for continuous-request scheduling, such as EDF and SCAN/EDF, affect the response time of discrete (or aperiodic, as they call them) requests. A simple scheduling scheme, called "the immediate server approach" [18] was employed for discrete requests, according to which discrete requests were served in between two successive EDF-selected continuous requests. In [11] the authors contribute an analysis of the trade-offs in managing the I/O bandwidth resource in a multimedia server with mixed-media workloads.

Since multimedia disk scheduling research has moved away from EDF-based algorithms, the work in [21] attempted to analytically model and predict the performance of multimedia servers with mixed workloads, when scheduling is based on the notion of rounds. The work in [21] is a preliminary attempt to define the issues and present initial algorithms

Many disk-scheduling algorithms consider mixed workloads in the sense of workloads with different Quality of Service (QoS) requirements and several algorithms have been developed that provide QoS guarantees to different classes of applications. Some representative efforts were made at [4,24,30]. In [30] the scheduler assigns weights to the application classes. Each class receives disk bandwidth proportional to its weight. In a similar approach in [4] an approximation of the Generalized Processor Sharing (GPS) scheduler is implemented and weights are assigned to the application classes according to the GPS. In [24] a two-level scheme is employed where bandwidth allocation and scheduling are separate tasks. The scheduling level cares only for the efficient scheduling of admitted requests and the bandwidth allocation level provides a proper share of the disk bandwidth to the different classes.

The later assumption is also made in our approach. We assume the existence of a separate task that delivers to the scheduler a proper number of requests per round in order the sessions to meet their QoS requirements. We focus on the scheduler, which is dedicated to efficiently schedule the delivered requests (from QoS sensitive applications) as well as best-effort requests. A preliminary version of this paper was published in [27]. Our work differs from all others and extends our previous work in [27] in that: (a) studies the performance of additional algorithms, (b) considers additional performance metrics, such as the fairness of the algorithms, (c) contributes novel algorithms and evaluates their performance showing significant further performance improvements, (d) compares the proposed hierarchical scheduling algorithms against flat scheduling algorithms, and (e) includes a study of how the proposed algorithms will perform on future disk products, using known technology improvement projections, which is very important given the pace with which related technology improves.

1.2. System model and problem definition

Our storage server receives a *mixed workload* consisting of requests for continuous and discrete objects. It employs an admission controller which bounds the number of admitted continuous objects, given the availability of resources (such as RAM buffers, disk bandwidth, etc.) [26]. The admission controller ensures that the service of the requests that are delivered to the scheduler is sufficient in order their performance requirements to be met. We also adopt the notion of scheduling with rounds, outlined above. In addition, the admission controller will reserve some sub round period for discrete requests which can be characterized as best-effort requests and are interested only for the response time and the fairness of the algorithms.

Within this framework, the problem at hand is to derive disk-scheduling algorithms that will meet the following performance goals:

1. Continuous objects meet their performance requirements. (The displays of continuous objects observe no glitches.)
2. Discrete requests have small average response times.
3. Discrete requests do not starve, and discrete requests are fairly served.

The disk's workload consists of N concurrent continuous-data requests, C-requests, per scheduling round and also discrete-data requests, D-requests. In each round, the desired continuous blocks are known before hand (they are simply the next blocks of the admitted streams). The requests for these blocks are in a waiting queue (C-queue). D-requests arrive according to a Poisson process with rate λ_D and enter their waiting queue (D-queue). In each round we aim to produce a schedule, which meets the above performance goals.

2. Mixed-load scheduling: algorithms and a taxonomy

Before proceeding to the description of the proposed algorithms we will define a taxonomy of related algorithms which we hope will be useful to designers, implementers, and researchers, in putting the problem and its prospective solutions in context. The taxonomy categorizes the algorithms along two key dimensions:

1. The number of separate *scheduling phases* in a round. The algorithms are categorized as Two-Phase Scheduling

(TPS) algorithms when two separate schedules are produced, one for the C-requests and one for the D-requests, and the phases are non-overlapping in time. One-Phase Scheduling algorithms (OPS) produce mixed schedules, containing both C- and D-requests. TPS algorithms are specified as TPS: $Phase1\_Alg/Phase2\_Alg$.

2. The number of *scheduling levels*. *Hierarchical or clustered scheduling* algorithms for D-requests will be presented. These algorithms are based on defining *clusters* of D-requests. At the higher level, the algorithms are concerned with the efficient scheduling of clusters. At the lower level, the algorithms are concerned with the scheduling of a cluster's requests. Hierarchical algorithms are specified as $(Alg\_High)/(Alg\_Low)$, where Alg_High and Alg_Low are the algorithms used at the higher and lower levels, respectively.

Consequently, using the above notation an algorithm specified as *TPS*: *ALG1*/[*ALG2/ALG3*] stands for a two phase scheduling algorithm that schedules C-requests using *ALG1* and D-requests using a hierarchical scheduling algorithm which forms clusters and serves clusters using *ALG2* and requests inside clusters using *ALG3*.

2.1. Clustering and hierarchical scheduling

We define a *C-interval* as the disk interval between the disk cylinders of any two requested C-fragments that are successive in the current SCAN's direction. All D-requests belonging to the same C-interval form a cluster. Hierarchical scheduling algorithms in essence decompose the problem of scheduling all D-requests into the sub problems of deriving efficient schedules of, first, the clusters and, then, each cluster's requests.

We propose two lower-level scheduling algorithms. Their motivation is as follows. Given a non-trivial set of C-requests, clustering results in a number of small C-intervals (such that any seek within the C-interval is a "short seek" [28]). Applying seek-optimizing algorithms (such as SCAN) in such small intervals (i) does not result in major savings, and (ii) focuses at the wrong overhead component. To give a concrete example, consider a disk drive with 6500 cylinders and a set of 15 requested C-fragments, randomly distributed over the disk's surface. Any seek within the defined 430-cylinder-wide C-intervals will be a "short seek", the cost of which, given current technology, is only a few (e.g., less than four) milliseconds. However, in such drives the rotational delay can be considerably higher than the seek cost (e.g., up to 6 msec, for the drives with the currently fastest rotating speed). Furthermore, the transfer time can actually be significantly higher than both of the above costs (e.g., a 200 KB image in a disk with 17 MB/sec transfer rate, requires a transfer time of 12 msec). Given, as mentioned in section 1, the capabilities of modern disk controllers, further optimizations, such as the ones that follow, are possible.

The CI-SPTF (C-Interval Shortest Positioning Time First) Algorithm
According to this algorithm:

1. The D-queue is partitioned into *interval queues*, one per C-interval. The D-requests are distributed to the appropriate interval queues.
2. Within each interval queue the D-requests are sorted[1] according to their total positioning costs (seek, and rotation times), in ascending order.
3. *CI-SPTF* serves D-requests according to their order in the interval queue.
4. The interval queue is resorted after the service of a D-request.

The CI-OPT (C-Interval Optimal) Algorithm
CI-SPTF is a greedy algorithm, which may not produce optimal schedules. *CI-OPT* looks at all D-requests in the interval queue, considers all possible schedules, and determines the one with the minimum cost. It is well known that such optimality problems are reducible to the Traveling Salesman Problem (TSP) [7,15]. For this reason, we place an upper bound on the number of cluster members (e.g., six, thus leading to the *CI-OPT*(6) algorithm). (Our studies have shown that the performance benefits for larger clusters become marginal, while the simulation run-times become intolerable.) According to this algorithm:

1. The interval queues are constructed as in *CI-SPTF*, except that when one is to contain more than the specified maximum of D-requests, it is recursively subdivided into two equal-sized queues.
2. The interval queues are then sorted in the order computed by *CI-OPT*.

The overall (hierarchical) algorithm serves the clusters of D-requests in some order (which in this paper is SCAN) and within each cluster (C-interval) serves the D-requests according to the order in the corresponding interval queue as produced by *CI-SPTF* or *CI-OPT*(6). Note that the performance results include this overhead.

In addition, before serving a D-request within a cluster, the algorithms check if there will be enough time to serve all remaining C-requests (using SCAN). If so, the D-request is served; else, it is postponed until the next round. Note that this check can rapidly be performed using the known detailed models for the seek cost, the rotational delays, and the transfer times.

2.2. Two phase scheduling

In two phase scheduling (TPS) schemes we serve C-requests and D-requests separately into disjoint time periods (C-period and D-period) within each round. Each schedule can be produced by any known or novel algorithm. In this paper we will assume that at the beginning of each round a schedule for the N C-requests is constructed according to the SCAN policy, and then in the remaining time (until the end of the current round) the D-requests are served according to either the SCAN discipline or a hierarchical scheduling discipline. Thus, we will focus on $\mathit{TPS}{:}\mathit{SCANSCAN}$ and on $\mathit{SCAN}/\frac{\mathit{SCAN}}{\mathit{CI\text{-}OPT}(6)}$, respectively. For the latter (hierarchical) algorithm, the clusters which will be visited in SCAN order during the second phase are defined as in the $\mathit{CI\text{-}OPT}(6)$ algorithm, except that instead of a C-interval, the whole disk is considered.

2.3. One phase scheduling

In one phase scheduling schemes C-requests and D-requests are served together in an interleaved manner. D-requests that arrive during the service period of C-requests, do not necessarily have to wait until the end of that period. We propose and study the following one-phase scheduling algorithms:

FAMISH (FAir MIxed-scan ScHeduling)
D-requests are ordered in the D-queue based on their arrival time. FAMISH ensures that:

- All C-requests will be served within the current round.
- No D-request D_i will be served in an earlier round than D_j, if D_j is ahead of D_i in the D-queue.

In particular, FAMISH will:

1. Construct a SCAN schedule for the C-requests for this round.
2. Incorporate the D-request at the head of the D-queue, in its proper position in the current SCAN schedule.
3. Calculate whether serving this SCAN schedule (which includes this D-request) will result in a hiccup for one or more C-requests in the SCAN.
4. If not, the iteration continues from step 2.
5. If so, the iteration stops, this D-request is removed from the current SCAN for this round and the requests in the resulting SCAN are served.

In addition to $\mathit{OPS}{:}\mathit{FAMISH}$, we will also consider the $\mathit{OPS}{:}\frac{\mathit{SCAN}}{\mathit{CI\text{-}SPTF}}$ and $\mathit{OPS}{:}\frac{\mathit{SCAN}}{\mathit{CI\text{-}OPT}(6)}$ algorithms.

3. Experimentation and performance results

3.1. Experimental testbed

Our simulation results have been extracted by modeling a modern disk drive. The parameters of the disk drive are given in table 1.

The seek time modeling as a function of the distance is given by

$$\mathrm{Seek}(d) = \begin{cases} 1.867 \cdot 10^{-3} + 1.315 \cdot 10^{-4}\sqrt{d}\ (\mathrm{msec}), \\ \quad \text{if } 1 \leqslant d \leqslant 1344, \\ 3.8635 \cdot 10^{-3} + 2.1 \cdot 10^{-6} d\ (\mathrm{msec}), \\ \quad \text{otherwise.} \end{cases} \tag{1}$$

Table 1
Disk characteristics.

Cylinders	6,720
Revolution speed	10,000 RPM
Rotation time	6 ms
Number of zones[a]	1
Transfer time	17 MBps
Track capacity	>90 Kbytes

[a] We assume 1 zone for simplicity.

Table 2
Data characteristics.

C-request size	Mean	200000 bytes
Gamma distributed	standard deviation	100000 bytes
D-request size	Mean	70000 bytes
Normal distributed	standard deviation	20000 bytes

The rotational delay for each D-request is computed by keeping track of the starting disk sector within the cylinder containing the D-request. Given the constant angular velocity of the disk and the seek distances, along with the accurate seek cost model above, we can calculate which sector will be passing beneath the disk head at the end of the seek and thus compute the actual rotational delay. The transfer time is given from the disk's transfer rate and the block size.

The data characteristics for C- and D-requests are given in table 2. Given that the round duration is 1 sec, the values for C-requests reflect typical data characteristics for MPEG-1 data. The sizes of D-requests are typically smaller and obey a normal distribution. Note that a good fraction of the D-requests are smaller than the track size. For these the rotational delay is a significant cost factor, whereas this effect dismisses for requests that are larger than a full track.

The arrival of D-requests is driven by a Poisson process with arrival rate λ_{D} and it is assumed that arriving D-requests are uniformly distributed over the disk. All simulations consist of five independent runs of 40,000 rounds each (which are enough for the algorithms to reach steady-state). During each run: the number of C-requests is held constant and the results for a specific performance metric are calculated with cumulative data. For example, the steady-state D-request average response time is obtained from the ratio of total response time of D-requests to the total number of serviced D-requests. The final value for a metric is calculated as the mean of that metric under the five runs.

3.2. Performance metrics

In order to evaluate the performance of the algorithms we consider the following metrics:

- *Mean response time* of the D-requests, which is the sum of the request waiting time (in a queue) and its service time. For a D-request the service time is the sum of the request's seek, rotation, and transfer time.
- *Fairness index*, which is given in equation (2) and it is analytically defined in [13]. This index is generally given by

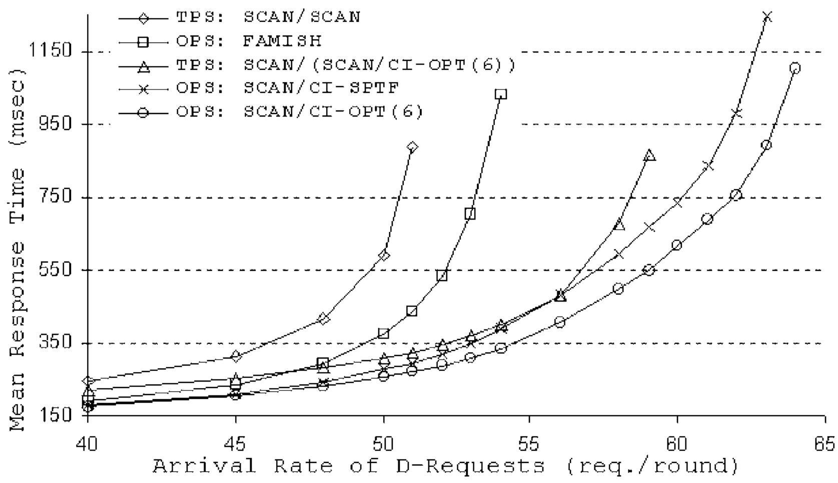


Figure 1. Mean response time for $N = 25$.

the formula $(E[X])^2/(E[X^2])$, where X is a random variable, and intuitively it is a further normalization than that in the squared coefficient of variation of X. Its usefulness is that it takes values between 0 and 1. An algorithm is fairer as its fairness index becomes closer to 1.

$$\text{Fairness index} = \frac{(\sum_{i=1}^{n} X_i)^2}{N \cdot \sum_{i=1}^{n} X_i^2}, \qquad (2)$$

where D_i is the response time of a single request and N is the number of serviced requests.

We note that the fairness index just like the mean response time are of no value for C-requests since the algorithms guarantee the performance requirement of continuous objects by just serving all C-requests in the round. However it is a very important performance metric when we consider D-requests which can be characterized as best-effort requests.

3.3. Performance comparison of algorithms

We compare the performance of the above scheduling algorithms. All algorithms prevent glitches (all C-requests are completed before the end of the round). The round's duration is set to 1 sec. In figure 1 we present the simulation results for the mean response time of D-requests for different arrival rates when $N = 25$. Serving 25 C-requests per round according to the SCAN algorithm and without intermixing C-requests in intervals between successive C-requests, consumes about 45% of the total round time.

Figure 1 shows that generally OPS algorithms outperform TPS algorithms except *OPS*:*FAMISH* which is focused in providing fairness among D-requests at the cost of response time. This result was expected since TPS algorithms force D-requests to wait until the end of the C-period. However we observe that *TPS*:*SCAN*/$\frac{SCAN}{CI\text{-}OPT(6)}$ performs significantly better than *TPS*:*SCAN*/*SCAN* and it's performance is comparable to OPS algorithms for low and medium D-request arrival rates. This emphasizes the usefulness of clustering and hierarchical scheduling. Among OPS algorithms *OPS*:$\frac{SCAN}{CI\text{-}OPT(6)}$ performs better than *OPS*:$\frac{SCAN}{CI\text{-}SPTF}$ as λ increases since in each cluster belong more D-requests and there is bigger difference between the optimal and the greedy schedule produced by the two algorithms, respectively.

In figure 2 we show the results for the fairness index. Due to the hierarchical nature of the algorithms, improvements in mean response time does not occur at the cost of fairness. While the lower level scheduler performs local optimizations in order to reduce latency, the higher-level scheduler favors fairness. *OPS*:*FAMISH* which is focused in providing fair schedules is as expected the fairer algorithm but for heavy workloads it leaves a considerable amount of D-requests without service at the end of each round. The general behavior of the algorithms remains the same when we consider different workloads (i.e., for $N = 15$ C-requests per round; we omit the presentation for space reasons).

4. Clustering properties and algorithms

4.1. Motivation

The performance results presented in the previous section clearly show the performance benefits that clustering introduces. In the hierarchical algorithms described in section 2 we constructed clusters using the notion of C-interval. In this section we study the effects of several clustering parameters and we develop novel clustering techniques that we later apply to the proposed hierarchical algorithms in order to improve their performance further.

4.1.1. Elements of a good clustering method

We investigate the importance of two factors in the construction of clusters. The first factor is the number of requests that a cluster contains and the second is the size of a cluster, measured in cylinders on the disk surface. We constructed several clusters with different sizes in requests and in cylinders and we measured the mean response time of the D-requests (a) using the OPT algorithm to serve requests within the cluster, and (b) using the SCAN algorithm to serve requests within the cluster. In the results presented in figure 3 we show the

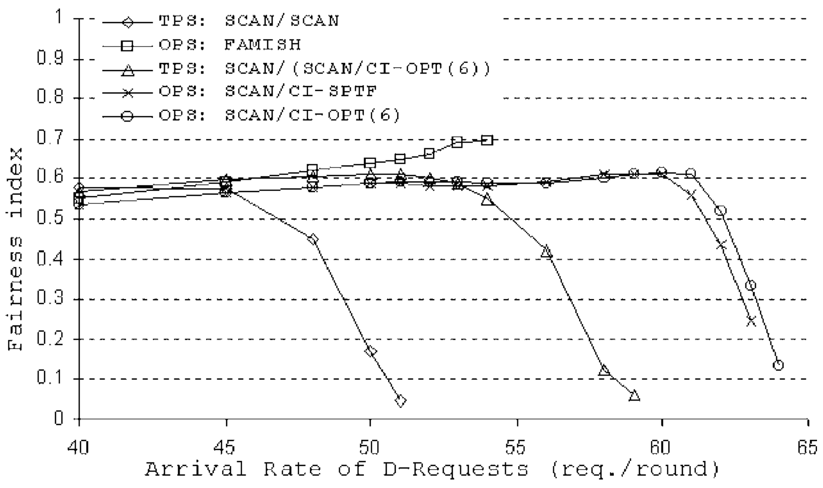


Figure 2. Fairness of algorithms for $N = 25$.

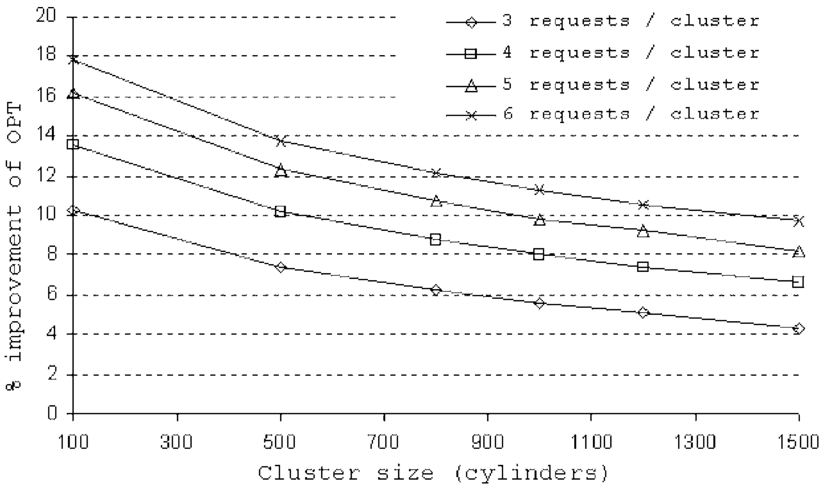


Figure 3. Efficiency of the OPT against SCAN.

% improvement in mean response time of the schedule produced by OPT against the schedule produced by SCAN.

For a specific cluster size in cylinders, we observe that performance improves as the number of requests that it contains increases. For a specific number of requests in a cluster, performance is better when the cluster size in cylinders is small. A good clustering method would be a method that would create compact clusters, i.e., small disk intervals with as many requests as possible.[2]

4.2. Alternative clustering techniques

Based on the above remarks we propose two clustering techniques, which attempt to improve the efficiency of the hierarchical scheduling algorithms presented in section 2, by trying to create compact clusters.

4.2.1. Clustered requests

We name the first technique *clustered requests*. According to this technique clusters are constructed as follows. A scan direction is initially determined and requests are inserted into the cluster until it contains a maximum number of requests (M requests). Parameter M is a design, tunable parameter. This technique is not concerned about the size of the cluster in cylinders but only about the number of requests that the cluster contains.

4.2.2. Clustered cylinders

We name the second technique *clustered cylinders*. Unlike the previous method where the size of the clusters (measured in cylinders) was completely ignored, in this clustering scheme it is an important design parameter. Particularly, in this scheme, requests are inserted into the cluster according to a scan direction until (a) the cluster extends up to a specific cylinder range (T cylinders), or (b) the cluster contains a maximum number of requests (M requests). Parameters T and M are the design parameters of this technique.

We will examine two algorithms that use the clustering techniques described above: The first is $OPS{:}\frac{SCAN}{CLUST\text{-}REQ(6)}$ which is a hierarchical scheduling algorithm that (a) constructs clusters according to "clustered requests" (b) serves clusters using SCAN, and (c) serves requests inside clusters using the OPT(6) algorithm. The second algorithm is $OPS{:}\frac{SCAN}{CLUST\text{-}CYL(6|1000)}$ which is similar with $OPS{:}\frac{SCAN}{CLUST\text{-}REQ(6)}$

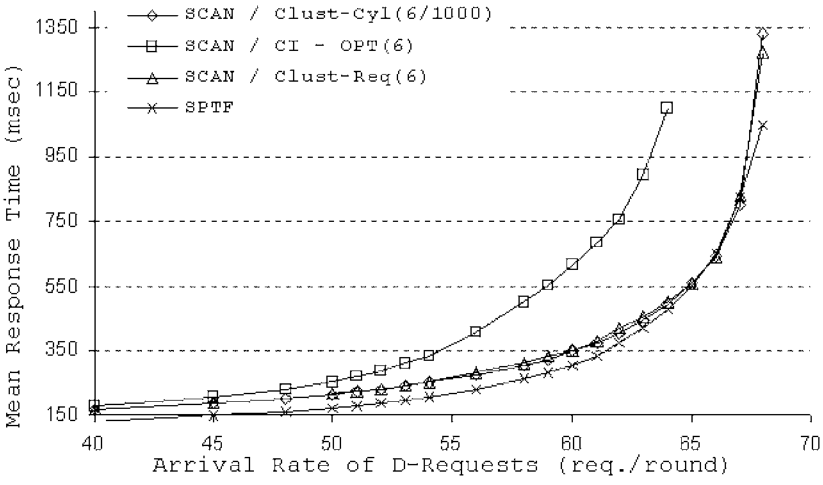


Figure 4. Mean response time for $N = 25$.

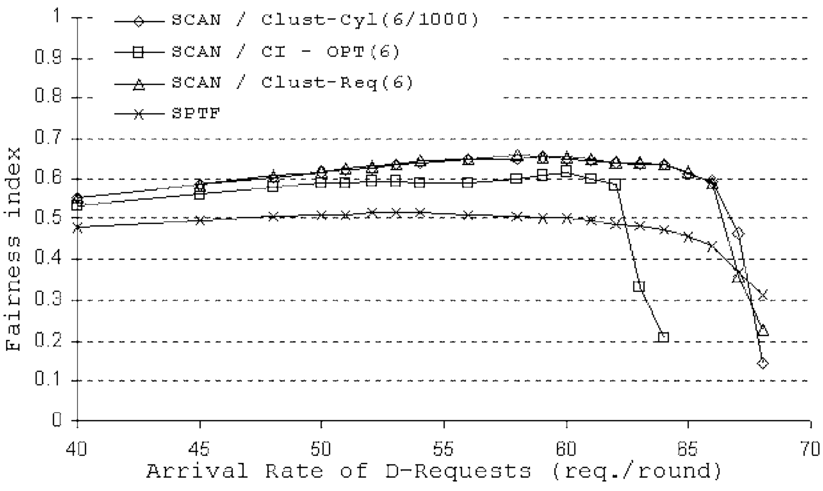


Figure 5. Fairness of algorithms for $N = 25$.

but constructs clusters according to "clustered cylinders" that are at most 1000 cylinders wide or they contain at most 6 requests.

4.3. Performance results

The disk and data characteristics are the same with these used in section 3. We compare all the proposed clustered scheduling algorithms against one another and against a traditional flat scheduling method SPTF (which we have found to perform best among all flat scheduling algorithms). In order to ensure that SPTF prevents glitches (i.e., that all C-requests are completed before the end of the round) we serve a D-request only if the service of this request leaves enough time to perform a SCAN for all the remaining C-requests. In figures 4 and 5 we present the simulation results for the mean response time of D-requests and the fairness index of the algorithms, respectively, for different arrival rates when $N = 25$ C-requests per round.

A first note that we can make looking at figures 4 and 5 is that $OPS{:}\frac{SCAN}{CLUST\text{-}REQ(6)}$ and $OPS{:}\frac{SCAN}{CLUST\text{-}CYL(6|1000)}$ have almost identical performance in both mean response time and fairness. This is not surprising as the compactness of the disk (which is the target of clustering) depends only on the arrival rate of requests and not on the two clustering techniques. Specifically, for high D-request arrival rate, requests are close to each other and parameter M is the one that determines the requests that are inserted in the clusters for both algorithms. Consequently both algorithms construct clusters using the same criterion thus they have similar performance. For low D-request arrival rates, the distance between requests is larger and "clustered requests" construct clusters with six requests spread in a relatively large disk interval. On the other hand "clustered cylinders" constructs small clusters with a relatively small number of requests. This means that for low D-request arrival rate, $OPS{:}\frac{SCAN}{CLUST\text{-}REQ(6)}$ calculates an inefficient (because of its large size in cylinders) optimal schedule for 6 requests while $OPS{:}\frac{SCAN}{CLUST\text{-}CYL(6|1000)}$ calculates a more efficient (because of it's small size) optimal schedule but for fewer requests.

A second important point is the benefits of the application of the new clustering techniques on our hierarchical algorithms. The reason for the significant improvement is that C-intervals, as described in section 2, are generally larger than

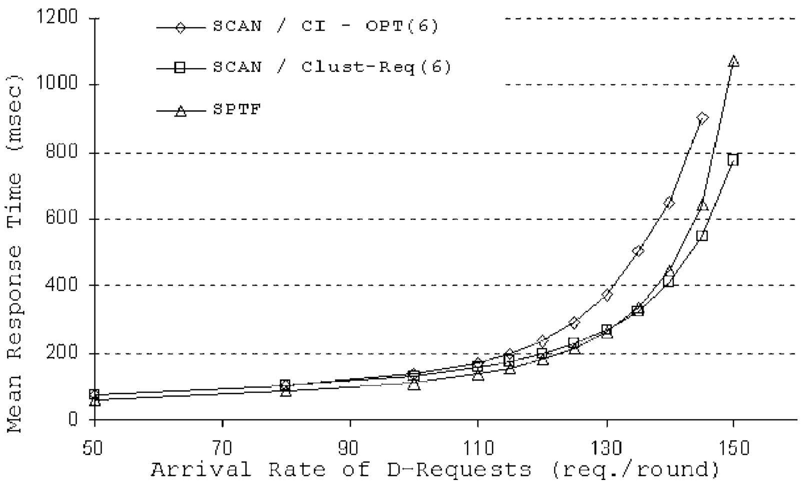


Figure 6. Mean response time for $N = 35$.

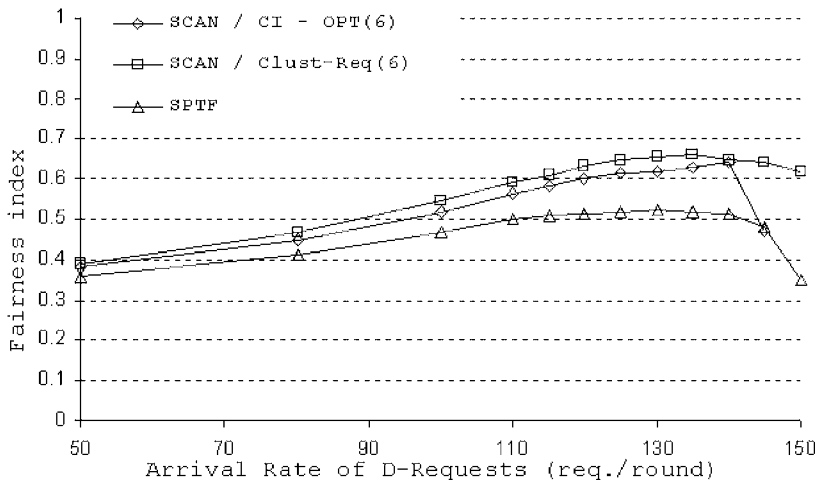


Figure 7. Fairness of algorithms for $N = 35$.

the clusters of the new simple techniques. As expected, the compactness achieved with the two new clustering techniques pays off.

Finally comparing the clustered scheduling algorithms with SPTF, we observe that generally SPTF has lower mean response time while the proposed clustered scheduling algorithms are fairer. However, it is important to note that the difference in the mean response time is not significant, especially for high D-request arrival rates. While in fairness, the proposed algorithms clearly outperform SPTF and become up to 30% fairer for high arrival rates (i.e., $\lambda = 65$ req./round). Unfairness is one of the main reasons that SPTF is not practically used by disk drive manufacturers. Our algorithms achieve to have mean response time comparable to that of SPTF while they are significantly fairer.

5. Sensitivity study on the rapidly developing disk technology

We now make a projection for 4 years of seek, rotational, and transfer time improvements in order to see how the algorithms are expected to perform in the near future. In our projection we assume 8% annual improvement for the seek cost, 20% for the rotational speed and at least 20% for the transfer rate. We also assume 50% annual improvement of CPU speed, which is related to the computational overhead for the computation of an optimal schedule in a cluster. In figures 6 and 7 we present the simulation results for $N = 35$ C-requests per round, which according to the SCAN algorithm corresponds to the 35% of the total round time.

Comparing figures 6 and 7 with figures 4 and 5 we observe an improvement in the performance, attributed to the new clustering techniques, which can be very significant in heavier loads. While the performance of the algorithms in mean response time is similar for low and medium workloads, hierarchical algorithms appear to be considerably fairer. For heavier loads the clustered scheduling algorithm outperforms the flat scheduling algorithm. Particularly $OPS{:}\frac{SCAN}{CLUST\text{-}REQ(6)}$ appears to have 15% lower mean response time than SPTF while it is 30% fairer. This drastic improvement should be expected because as the system's load becomes heavier the compactness of the clusters increases and there is a better chance for the local optimization to work efficiently.

6. Conclusions

We have considered the problem of disk scheduling for mixed-media workloads. Despite the fact that such workloads are typical of many applications, this problem has not received the attention it deserves by related research efforts. We have presented several algorithms, which aim to ensure the hiccup-free display of continuous objects, while ensuring low average response times and fairness for discrete data requests. We have contributed a taxonomy of related algorithms, organized along two key dimensions: the number of separate scheduling phases and the number of scheduling levels.

We have implemented the proposed algorithms in a detailed simulation testbed. We have shown that the hierarchical scheduling algorithms we propose can offer drastic improvements, up to several hundred percent, in the average response times of D-requests. Also, these impressive response times do not occur at the cost of unfair D-request schedules. The key to reconciling the goals of achieving response time optimizations and fair schedules is the hierarchical nature of the proposed algorithms. At the higher level, the algorithm with which the clusters are served can be chosen to account for fairness and efficiency. At the lower level, algorithms can be chosen which allow the local optimization of response times.

Another key contributing factor to the success of the proposed algorithms is the selection of appropriate clustering methods, which define the clusters. We proposed simple clustering methods, which improve the response time behavior significantly while also improving fairness.

Finally we have studied the performance of the algorithms in future disk drives, using standard technology projections. This study shows that the proposed clustered scheduling algorithms are expected to perform comparatively even better in the future, outperforming flat scheduling algorithms in both response times and fairness.

Ongoing work includes the design and evaluation of alternative higher-level scheduling algorithms and efficient heuristics for TSP which can avoid the upper bound of six D-requests per cluster and replace the OPT(6) algorithm. Also, we plan to investigate to what extent similar hierarchical algorithms, in which the higher-level algorithms effectively avoid fairness problems introduced by the lower-level optimizations, are suitable for file-system or database-system disk workloads.

Acknowledgements

This research has been supported by the EXAPSIS project funded by the General Secretary of Research and Technology, Greece.

Notes

1. Sorting can be omitted in the implementation in order to reduce the complexity of the algorithm. It is mentioned because it simplifies the description of the algorithm.
2. The use of the OPT method on clusters puts an upper bound to the number of requests that a cluster can contain.

References

[1] S. Berson, S. Ghandeharizadeh, R.R. Muntz and X. Ju, Staggered striping in multimedia information systems, in: *Proc. of the Int. Conference on Management of Data (SIGMOD)*, Minneapolis, MN, 1994, pp. 79–90.

[2] V. Bharadwaj, D. Ghose, V. Mani and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems* (IEEE Computer Society Press, 1996).

[3] W.J. Bolosky, J.S. Barrera, III, R.P. Draves, R.P. Fitzgerald, G.A. Gibson, M.B. Jones, S.P. Levi, N.P. Myhrvold and R.F. Rashid, The Tiger Video Fileserver, MSR-TR-96-09, ftp://ftp.research.microsoft.com/pub/tr/tr-96-09.ps

[4] J. Bruno, E. Gabber, B. Ozden and A. Silberschatz, Disk scheduling algorithms with quality of service guarantees, in: *Proc. of the IEEE Conference of Multimedia Computing Systems (ICMCS'99)*, June 1999.

[5] Coffman, Jr. and M. Hofri, Queueing models of secondary storage devices, in: *Stochastic Analysis of Computer and Communication Systems*, ed. H. Takagi (North-Holland, 1990).

[6] P.J. Denning, Effects of scheduling on file memory operations, in: *Proc. AFIPS Conference*, April 1967, pp. 9–21.

[7] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, New York, 1979).

[8] R. Geist and S. Daniel, A continuum of disk scheduling algorithms, ACM Transactions on Computer Systems (February 1987) 77–92.

[9] D.J. Gemmel, H.M. Vin, D.D. Kandlur, P.V. Rangan and L.A. Rowe, Multimedia storage servers: A tutorial, IEEE Computer 28(5) (1995) 40–49.

[10] S. Ghandeharizadeh, S.H. Kim and C. Shahabi, On disk scheduling and data placement for video servers, ACM Multimedia Systems (1996).

[11] L. Golubchik, J.C.S. Liu, E. de Silva e Souza and H.R. Gail, Evaluation of tradeoffs in resource management techniques for multimedia storage servers, CS-TR-3904, University of Maryland, May 1998.

[12] D. Jacobson and J. Wilkes, Disk scheduling algorithms based on rotational position, Techical Report HPL-CSP-91-7, February 1991.

[13] R. Jain, D.-M. Chiu and W.R. Hawe, A quantitative measure of fairnsess and discrimination for resource allocation in shared computer systems.

[14] A. Laursen, J. Olkin and M. Porter, Oracle media server: Providing consumer based interactive access to multimedia data, in: *ACMSIGMOD'94* (1994) pp. 470–477.

[15] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (Wiley, New York, 1986).

[16] K. Li, Managing divisible load on a partionable network, in: *High Performace Computing Systems and Applications*, ed. J. Schaeffer (Kluwer Academic Publishers, 1998) pp. 217–228.

[17] X. Li, V. Bharadwaj and K.C. Chung, Divisible load scheduling with start-up costs on distributed linear networks, in: *12th Int. Conference on ISCA PDCS*, Florida, USA, August 1999.

[18] T.H. Lin and W. Tarng, Scheduling periodic and aperiodic tasks in hard real time computing systems, in: *Proc. Int. ACM SIGMETRICS Conference* (1991).

[19] C.L. Liu and J.W. Layland, Scheduling algorithms for multiprogramming in a hard real time environment, J. ACM 20(1) (1973) 46–61.

[20] C. Martin, P. Narayan, B. Ozden, R. Rastogi and A. Silberschatz, The Fellini multimedia storage system, in: *Multimedia Information Storage and Management*, ed. S.M. Chung (Kluwer Academic Publishers, 1996) ch. 5.

[21] G. Nerjes, P. Muth, M. Paterakis, Y. Romboyannakis, P. Triantafillou and G. Weikum, Incremental scheduling of mixed workloads in multimedia information servers, International Journal on Multimedia Tools and Applications (2000) (an earlier version appeared in *The Proc. IEEE Int. RIDE Workshop*, February 1998).

[22] B. Ozden, R. Rastogi and A. Silberschatz, Disk striping in video server environments, in: *Proc. of the Int. Conference on Multimedia Computing and Systems (ICMCS)*, June 1996.

[23] Quantum Corp., Storage Basics, ORCA, www.quantum.com/src/storage/_basis.
[24] A. Reddy and J. Wyllie, Intergraded QoS Management for disk I/O, in: *IEEE Conference. on Multimedia Computing and Systems*, June 1999.
[25] A.L.N. Reddy and J.C. Wyllie, I/O issues in a multimedia system, IEEE Computer (March 1994) 69–74.
[26] M. Reisslein, F. Hartanto and K.W. Ross, Interactive video streaming with proxy servers (extended version), Technical Report, GMD FOKUS, June 1999.
[27] Y. Romboyannakis, G. Nerjes, P. Muth, M. Paterakis, P. Triantafillou and G. Weikum, Disk scheduling for mixed-media workloads in multimedia servers, in: *Proc. ACM Multimedia Conference*, Bristol, UK, 1998.
[28] C. Ruemmler and J. Wilkes, An introduction to disk drive modeling, IEEE Computer (March 1994) 17–28.
[29] M. Seltzer, P. Chen and J. Ousterhout, Disk scheduling revisited, in: *Proc. USENIX Technical Conference* (1990) pp. 313–323.
[30] P. Shenoy and H. Vin, Cello: A disk scheduling framework for next generation operating systems, in: *Proc. SIGMETRICS'98*, ACM, June 1998.
[31] F. Tobagi, J. Pang, R. Baird and M. Gang, Streaming raid – a disk array management system for video files, in: *1st ACM Conference on Multimedia*, August 1993.
[32] P. Triantafillou and C. Faloutsos, Overlay striping and optimal parallel i/o in modern applications, Parallel Computing 24 (March 1998) 21–43.
[33] P.S. Yu, M.S. Chen and D.D. Kandlur, Grouped sweeping scheduling for DASD-based multimedia storage management, ACM Multimedia Systems 1(3) (1993) 99–109.

Elias Balafoutis received the diploma degree in electronics and computer engineering from Technical University of Crete, Greece in 2000. Currently he is a graduate student at the Department of Informatics, University of Athens in the filed of Communications Systems and Networks and a member of the Communication Networks Laboratory. He has participated in several National and European research projects. His interests are in the area of multimedia systems and applications and particularly in Multimedia content distribution and retrieval.

Michael Paterakis received his diploma degree from the National Technical University of Athens, Greece, his M.Sc. degree from the University of Connecticut, and his Ph.D. degree from the University of Virginia, in 1984, 1986, and 1988, respectively, all in electrical engineering. Since 1995, he is a faculty member in the Department of Electronic and Computer Engineering (ECE) at the Technical University of Crete, Greece, where he is currently a Professor and Director of the University's Telecommunication Systems Research Institute. During September 1999–August 2001, he served as the Chairman of the ECE Department. He was an Associate Professor in the Department of Computer and Information Sciences (CIS) at the University of Delaware, on the faculty of which he has been since September 1988.

His research interests include computer communication networks with emphasis on protocol design, modeling and performance evaluation of broadband high speed networks, of multiple access wireless microcellular communication systems, and of packet radio networks; centralized and distributed multimedia information delivery systems; queueing and applied probability theory and their application to computer communication networks and information systems. He has published extensively (over 75 papers) in archival scientific journals, refereed conference proceedings and edited books, in the abovementioned technical areas. He was invited to serve on the Technical Program Committees of the 1991 Int. Conference on Distributed Computing Systems, the 1992 and 1994 IEEE INFOCOM Conferences, the 1997 IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'97), the 6th IFIP Workshop on Performance Modeling and Evaluation of ATM Networks (IFIP ATM'98), and of the 2001 Very Large Databases Conference (VLDB'01). He has served as reviewer for almost all of the major IEEE Transactions and other international archival scientific/technical journals and for most of the major international conferences in his research areas.

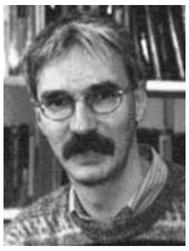
Professor Paterakis is a Senior Member of the IEEE. He is also a member of the IEEE Technical Committee on Computer Communications, the Greek Chamber of Professional Engineers, and the Greek Association of Electrical and Electronic Engineers.

Peter Triantafillou was born in Toronto Canada in 1963. He received the Ph.D. degree from the Department of Computer Science at the University of Waterloo in 1991. Until August 1996 he was an Assistant Professor at the School of Computing Science at Simon Fraser University, Canada. From September 1994 till January 1996 he was on a leave of absence at the Department of Electronic and Computer Engineering at the Technical University of Crete, Greece. Since January 1996, he has been with the same department. Currently Peter Triantafillou is a Full Professor and Director of the Software Systems Engineering and Network Applications (SOFTNET) Laboratory.

Professor Triantafillou's research efforts currently focus on Internet Content Delivery and Integration. In the recent past he worked on intelligent storage systems and on distributed servers, with emphasis on supporting multimedia applications. Earlier research activities had focused on multidatabases, distributed file systems, and highly-available distributed databases. Professor Triantafillou has published extensively in all the above areas and has been invited to serve in several Program Committees of related international conferences and as a reviewer in most relevant international journals and conferences.
E-mail: peter@softnet.tuc.gr

Guido Nerjes received his diploma degree (Dipl.-Inform.) from the Technical University of Braunschweig, Germany, in 1996 and his doctoral degree (Dr.-Ing.) from the University of the Saarland at Saarbruecken, Germany, in 2000. He is currently a system engineer with an advanced technology group of Bertelsmann. From 1996 through 1998 he was affiliated with the Institute of Information Systems at ETH Zurich, Switzerland, where he was involved in the Esprit research project Hermes. His research interests include performance analysis, multimedia information systems, and Web applications.

Peter Muth received his diploma degree (Dipl.-Inform.) and doctoral degree (Dr.-Ing.) both from the University of Darmstadt, Germany, in 1989 and 1994, respectively. He is currently affiliated with an insurance company, Berlinische Leben AG, in Wiesbaden, Germany. Until 1994 he was leading a department in the Integrated Publication and Information Systems Institute (IPSI) of the National Research Center for Information Technology (GMD) in Darmstadt. From 1994 through 1998 he was an Assistant Professor in the Department of Computer Science of the University of the Saarland at Saarbruecken, Germany. Dr. Muth's research interests include parallel and distributed information systems, multimedia information systems, and workflow management.



Gerhard Weikum received the diploma degree (Dipl.-Inform.) and the doctoral degree (Dr.-Ing.) both in computer science from the University of Darmstadt, Germany, in 1982 and 1986, respectively. Dr. Weikum is a Full Professor in the Department of Computer Science of the University of the Saarland at Saarbruecken, Germany, where he is leading a research group on database systems. His former affiliations include MCC at Austin, Texas, and ETH Zurich in Switzerland. During his sabbatical in 1997, he was a visiting Senior Researcher in the database research group of Microsoft. Dr. Weikum's research interests include parallel and distributed information systems, transaction processing and workflow management, and database optimization and performance evaluation. Dr. Weikum serves on the editorial boards of ACM Transactions on Database Systems, The VLDB Journal, and the Distributed and Parallel Databases Journal. He has served on numerous program committees of international conferences, he was the Program Committee Co-Chair of the 4th International Conference on Parallel and Distributed Information Systems, and he is the Program Committee Co-Chair of the 16th IEEE International Conference on Data Engineering. Dr. Weikum has been elected onto the board of trustees of the VLDB Endowment.