Brief Contributions

Random Redundant Storage in Disk Arrays: Complexity of Retrieval Problems

Joep Aerts, Jan Korst,
Frits Spieksma, Wim Verhaegh, and
Gerhard Woeginger

Abstract—Random redundant data storage strategies have proven to be a good choice for efficient data storage in multimedia servers. These strategies lead to a retrieval problem in which it is decided for each requested data block which disk to use for its retrieval. In this paper, we give a complexity classification of retrieval problems for random redundant storage.

Index Terms—Random redundant storage, load balancing, video servers, complexity analysis.

1 Introduction

A multimedia server [13] offers continuous streams of multimedia data to multiple users. In a multimedia server, one can generally distinguish three parts: an array of hard disks to store the data, an internal network, and fast memory used for buffering. The latter is usually implemented in random access memory (RAM). The multimedia data is stored on the hard disks in blocks such that a data stream is realized by periodically reading a block from disk and storing it in the buffer, from which the stream can be consumed in a continuous way. A block generally contains a couple of hundred milliseconds of video data. The total buffer space is split up into a number of buffers, one for each user. A user consumes, possibly at a variable bit rate, from his/her buffer and the buffer is repeatedly refilled with blocks from the hard disks. A buffer generates a request for a new block as soon as the amount of data in the buffer becomes smaller than a certain threshold. We assume that requests are handled periodically in batches, in a way that the requests that arrive in one period are serviced in the next one [16]. In the server, we need a cost-efficient storage and retrieval strategy that guarantees, either deterministically or probabilistically, that the buffers do not underflow or overflow.

Load balancing is very important within a multimedia server, as efficient usage of the available bandwidth of the disk array increases the maximum number of users that can be serviced simultaneously, which results in lower cost per user. Random redundant storage strategies have proven to enable a good load balancing performance [1], [3], [15], [23]. In these storage strategies, each data block is stored more than once, on different, randomly chosen disks. This data redundancy gives the freedom to obtain a balanced load with high probability. To exploit this freedom, an algorithm is needed to solve, in each period, a retrieval problem, i.e., we have to decide, for each data block, from which disk(s) to retrieve it in such a way that the load is balanced.

- *J. Aerts, J. Korst, and W. Verhaegh are with Philips Research Laboratories, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands. E-mail: aertsjcqm.nl, {jan.korst, wim.verhaegh}@philips.com.*
- *F. Spieksma is with Katholieke Universiteit Leuven, Naamsestraat 69, 3000 Leuven, Belgium. E-mail: frits.spieksma@econ.kuleuven.ac.be.*
- *G. Woeginger is with Universiteit Twente, PO Box 217, 7500 AE Enschede, The Netherlands. E-mail: g.j.woeginger@math.utwente.nl.*

Manuscript received 17 May 2001; revised 7 May 2002; accepted 11 Sept. 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 114166.

In this paper, we consider two load balancing approaches. In block-based load balancing, we balance the number of block requests assigned to the disks, whereas, in time-based load balancing, we balance the time that each disk needs for the retrieval of the assigned block requests. The block-based approach is the more conventional load balancing approach, whereas the time-based approach was introduced by the authors in [3]. Time-based load balancing has the advantage that it can also be applied in case of heterogeneous streams and heterogeneous disks. Furthermore, the multizone character of disks can be exploited, which leads to more efficient usage of the disk array [4], in a way that more blocks are read from the faster, outer zones of the disks. In this paper, we analyze the time complexity of the retrieval problems for block-based as well as time-based load balancing. For an elaboration on the load balancing algorithms, we refer to [1], [3], [15], [23]. The remainder of the paper is organized as follows: In the next section, we discuss related work in the area of multimedia servers. In Section 3, we model the block-based and the time-based retrieval problem and explain the relation to the multiprocessor scheduling problem [19] that is defined in the field of combinatorial optimization. We deal with the complexity of the block-based retrieval problem in Section 4 and with the complexity of the time-based retrieval problem in Section 5. In Section 6, we give an overview of the complexity results.

2 Related Work

Several papers describe the implementation of multimedia storage servers, such as those describing the PRESTO multimedia storage network [5], the MARS project [8], and the RIO project [24].

Most papers propose disk striping strategies to distribute the video data over the disks. Berson et al. [7], Chua et al. [9], and Santos et al. [25] describe data striping techniques. However, these striping techniques have some disadvantages, especially when used for variable-bit-rate streams. Most striping techniques distribute the consecutive blocks of a video file in a round-robin fashion over the disks of the disk array. This storage strategy is especially suited for constant-bit-rate streams, but results in large waiting times when the system is highly loaded. An alternative striping strategy is to split up each data block into a number of subblocks and request these subblocks in parallel. If we would use as many subblocks as the number of disks, a request for a block results in a request for a subblock on each disk such that a perfect load balance is guaranteed. Disadvantages of this strategy are that, in case the bandwidth requirements are the bottleneck, the total buffer size grows quadratically in the number of users and that the switch overhead increases due to the increase in the number of requests [13].

Korst [15] and Santos et al. [25] show that, for variable bit-rates and less predictable streams, e.g., in case of MPEG encoded video or VCR functionality, random multiplicated storage strategies outperform the striping strategies. In these strategies, each block is stored on a randomly chosen disk and (for some of the blocks) a copy is stored on another randomly chosen disk. Korst discusses random duplicated assignment and describes algorithms that balance the number of block requests assigned to each disk in each period. In his approach, he does not take into account the actual transfer times of the blocks. Santos et al. describe an asynchronous disk system in which they use shortest queue scheduling to assign the block requests to the disks. However, they use a FIFO disk scheduling algorithm, whereas a SCAN-approach would decrease the switch overhead. Both papers use replication schemes, but do not discuss how to exploit the multizone character of the disks.

Sanders describes alternative online scheduling strategies for asynchronous disk control in [21] and other scenarios, such as disk failures and scheduling of variable size requests in [22]. Aerts et al. [1], Sanders et al. [23], and Berenbrink et al. [6] prove that, with high probability, random duplicated storage results in a good load balance. Aerts et al. [3] and Sanders [22] show that the time-based retrieval problem is NP-complete, but none of the above papers focuses on the computational complexity of the retrieval problem. The aim of this paper is to give a complexity overview.

3 Problem Modeling

In each period, the following retrieval problem has to be solved. Given is a set $J = \{0, \ldots, n-1\}$ of blocks that have to be retrieved from a set $M = \{0, \ldots, m-1\}$ of hard disks. We want to select, for each block, the disk from which it is to be retrieved such that the load of the disks is balanced. In block-based load balancing, we take as optimization criterion the maximum load, i.e., the maximum number of block requests assigned to one of the disks. This results in the following retrieval problem.

Problem 1 (Block-based retrieval problem (BRP)). Given are n blocks that have to be retrieved from m disks and, for each block j, the set M_j of disks on which it is stored. Select, for each block j, a disk out of M_j in such a way that the maximum number of blocks to be read from any disk is minimized. The feasibility variant of BRP is defined as the question whether or not an assignment of block requests to disks exists with a maximum load of at most K.

In time-based load balancing, we minimize the time at which the last disk finishes the retrieval of its assigned blocks. The completion time of a disk equals the sum of the retrieval times of the blocks plus the total switch time, where the switch time consists of seek times and rotational delays. Seek time is the time required to move the disk head from one track to another. We approximate the seek time with a function linear in the distance that the disk head has to move. For the rotational delay, we use the worst-case assumption of one rotation per disk access. Furthermore, we assume that the disk head has to move in each cycle entirely from inside to outside or vice versa to retrieve the assigned block requests. Then, we can compute the total switch time per disk by a function linear in the number i of blocks assigned to the disk, i.e., the switch time equals $s \cdot i + c$ with $s, c \geq 0$.

The retrieval time of a block depends on the location of the block on the disk. Disks consist of multiple zones [20], where the outer zones have a higher transfer rate than the inner zones, but the transfer rate within a zone is constant. The information of the zone location of blocks on disks is assumed to be available, so the retrieval time of each block is known beforehand. The decision on how to distribute the blocks over the zones is defined by the storage strategy [3] and is considered to be beyond the scope of this paper.

In contrast to the block-based retrieval problem, we allow in the time-based retrieval problem that blocks are partially retrieved from different disks as long as each block is fetched completely. In this way, there is more freedom for load balancing. The drawback of splitting up a block access is that the total number of accesses increases, which results in a larger total switch time. We can define the time-based retrieval problem as follows.

Problem 2 (Time-based retrieval problem (TRP)). Given are n blocks that have to be retrieved from m disks and, for each block j, the set M_j of disks on which it is stored. Furthermore, the retrieval times of the blocks and the parameters s and c of the linear switch time function are given. The problem is to assign (fractions of) each block request j to the disks of M_j such that:

- Each block is fetched entirely and
- The completion time of the disk that finishes last is minimized, where the completion time equals the sum of the total switch time and total transfer time.

The feasibility variant is defined as the question whether or not an assignment of block requests to the disks exists that finishes at, or before, a given time T.

Retrieval problems can be seen as a special class of multiprocessor scheduling problems by viewing the disks as machines and the requested blocks as jobs. The transfer time of block j corresponds to the processing time p_j in the scheduling problem and the switch time corresponds to a set-up time. We will give a short introduction into the three-field notation of scheduling problems and, afterward, model BRP and TRP in this notation. For a more elaborate discussion of the three-field notation, we refer to [19].

In the three-field notation, the first field gives the machine environment, the second one describes the job characteristics, and the third one the optimization criterion. In the retrieval problems, we have parallel machine environments, indicated by P or R, corresponding to parallel identical machines and unrelated machines, respectively. The difference between P and R is that, in the case of P, the processing time p_j of a job j is equal on all machines, whereas, in the case of R, the processing time p_{ij} of job j also depends on the machine i. To indicate that we have a fixed number m of parallel identical machines, we use Pm.

For the second field, we introduce four job characteristics that are necessary for the retrieval problems.

- When we have unit processing times, as is the case in BRP, we indicate this by "$p_j = 1$."
- Machine eligibility is denoted by "M_j," which means that only machines of subset M_j are available for job j.
- Set-up times, denoted by "set-up," indicate that we need a certain time to set up the machine before starting a new job. In the retrieval problems, this is the switch time.
- Preemption, denoted by "pmtn," indicates that we allow job splitting. In the retrieval problem, this means that we allow that a block is partially retrieved from different disks. Preemption in the retrieval problem is not exactly the same as preemption in the general scheduling literature as we allow that multiple disks retrieve parts of one block at the same time. We use "pmtn^*" to denote this generalized variant of preemption.

As optimization criterion, we will only use $C_{\max}$, i.e., the completion time of the machine that finishes last. It equals the completion time of the last job and is referred to as the makespan.

Now, we can formulate BRP and TRP as multiprocessor scheduling problems. In BRP, we want to minimize the maximum number of blocks assigned to any disk. This means that we have jobs with unit processing times in a parallel identical machine environment and makespan as optimization criterion. Furthermore, we have machine eligibility constraints as, for each job, only a subset of the machines can be used. In the three-field notation, this problem can be denoted by $P|M_j, p_j = 1|C_{\max}$.

For TRP, the machine environment is given by unrelated parallel machines as a block need not be stored in the same zone for all disks on which it is stored. Again we have machine eligibility as a job characteristic. Furthermore, we have a set-up time for each job. This set-up time is constant as we approximate the total switch time with a linear function. To enable partial retrieval, we allow generalized preemption. The optimization criterion is again the makespan. Hence, in the three-field notation, TRP can be denoted by $R|M_j, \mathrm{pmtn}^*, \text{set-up}|C_{\max}$.

In this paper, we derive time complexity results for a number of retrieval problems, block-based as well as time-based. Fig. 2 gives an overview of the results that are described in this paper. In this

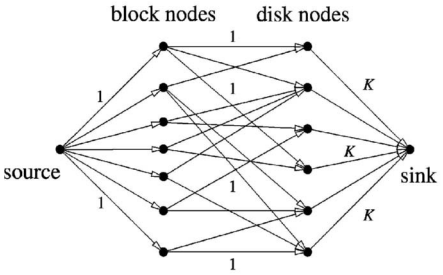

Fig. 1. Example of a max-flow graph for feasibility variant of BRP.

paper, we only give sketches of the proofs of the complexity results. We refer to [2] for the complete proofs.

4 Block-Based Load Balancing

In this section, we discuss the time complexity of solving the block-based retrieval problem. We can solve BRP for all multiplicated storage strategies with a max-flow algorithm. We define a directed max-flow graph for BRP as follows: The set of nodes consists of a source s, a sink t, a node for each disk, and a node for each block to be retrieved. The set of arcs consists of 1) arcs with unit capacity from the source to each block node, 2) arcs with unit capacity from each block node j to the disk nodes corresponding to the disks of M_j, and 3) arcs with capacity K from each disk node to the sink. An example of such a max-flow graph is shown in Fig. 1.

With this max-flow graph, we can solve the feasibility variant of BRP. Recall that networks with integral capacities admit maximal flows that are integral in all edges. If an integral maximum flow from source to sink saturates all edges leaving the source, then this flow corresponds to a feasible assignment of blocks to disks. This means that this solution approach not only solves the feasibility question, but also gives an assignment, in case of a positive answer, which can be derived from the flow over the arcs between the block nodes and the disk nodes.

Theorem 1. *Using the Dinic-Karzanov maximum flow algorithm [10], [18], the feasibility variant of BRP can be solved in $O(mn)$ time and the optimization variant in $O(\min\{mn\log n, m^2n, n^2\})$ time, assuming that the size of each set M_j is bounded by a constant.*

Proof (sketch). For the complete proof, we refer to [2]. The proof follows the lines of the complexity result of $O(|V|^3)$ for the Dinic-Karzanov algorithm for general graphs [18]. We strengthen this result by using the bipartite structure of the BRP graph, which results in a longest path of length $O(m)$ instead of length $O(|V|)$, and by using the fact that all arcs between block nodes and disk nodes have unit capacity. □

Theorem 2. *Using the preflow-push maximum flow algorithm [14], the feasibility variant of BRP can be solved in $O(mn)$ time. The optimization variant can also be solved in $O(mn)$ time by following the parametric approach [11]. Both statements hold under the assumption that the size of each set M_j is bounded by a constant.*

Proof (sketch). For the proof of the result for the feasibility variant, we again use the bipartite structure and the unit capacities of the BRP graph. For the optimization variant, we model BRP as a parametric max-flow problem in which K is the parameter. By reversing the arcs in the graph of Fig. 1, it fits in the constraints of Gallo et al. [11] to apply the parametric maximum flow algorithm for which they show that the optimization variant can be solved in the same time complexity as the feasibility variant. □

For practical situations, the assumption that the size of each set M_j is bounded by a constant is not a restriction as the maximum multiplication factor in any relevant storage strategy is always bounded by a constant. Besides, the size of the sets M_j is always bounded by m such that the time complexity bounds for the maximum flow algorithms grow at most with a factor m, if the assumption would not hold.

In case of duplicate storage, i.e., $|M_j| = 2$ for all blocks, Korst [15] describes an alternative graph formulation that gives a fourth time complexity bound of $O(n + m^3 \log n)$.

5 Time-Based Load Balancing

This section discusses the time complexity of TRP. We note that all variants of TRP that we discuss are problems with constant set-up times as we assume a linear switch time function. In case we do not allow preemption, the switch time parameter s can be included in the transfer times, which means that the complexity of a problem does not change by introducing only a set-up time. In case of preemption, however, the set-up times complicate the problems.

We first prove that the feasibility variant of TRP is NP-complete in the strong sense by a reduction from 3-partition, which is NP-complete in the strong sense [12].

Problem 3 (3-Partition). Given are a set of integers $A = \{a_1, \ldots, a_{3k}\}$ and a bound B, for which $\frac{B}{4} < a_i < \frac{B}{2}$ for all i and $\sum_i a_i = kB$. The question is whether or not A can be partitioned into k subsets such that the sum of the elements of each subset equals B.

Theorem 3. *The feasibility variant of TRP, denoted by $R|M_j, \mathrm{pmtn}^*, \text{set-up}|C_{\max} \leq T$ is NP-complete in the strong sense.*

Proof. It is obvious that we can check in polynomial time for a given assignment whether or not all disks are finished at time T, so the problem is in NP. To show that the problem is NP-complete, we show that a polynomial time reduction from 3-partition to TRP exists.

Considering an instance of 3-partition, we construct an instance of TRP in the following way: We take k disks and define, for each number a_j of the 3-partition instance, a block j, which is stored on all disks and has a transfer time $p_{ij} = p_j = a_j$ on each disk i. Furthermore, we define the time bound T of TRP as $T = 4B$ and the values s and c of the switch time function as B and 0, respectively. Now, we show that a positive answer for 3-partition is equivalent to a positive answer for TRP.

$\Rightarrow$ Given a solution to the 3-partition instance, we assign each subset to a different disk. For each disk, the sum of the transfer times equals B and three times a set-up time is needed, so the completion time for each disk equals $4B$.

$\Leftarrow$ Assume we have an assignment for TRP with value $4B$. As the transfer times are strictly larger than zero and $s = B$, no disk retrieves more than three blocks and no blocks are preempted. Consequently, each disk retrieves exactly three blocks. Combining this with the facts that $\sum p_i = kB$ and no disk exceeds $4B$, we conclude that the blocks assigned to each disk form a feasible subset in 3-partition. This proves that $R|M_j, \mathrm{pmtn}^*, \text{set-up}|C_{\max} \leq T$ is NP-complete in the strong sense. Note that this also proves that $P|M_j|C_{\max} \leq T$ is NP-complete in the strong sense as preemption is not used and, consequently, the set-up times can be included in the transfer times. □

The next theorem states that TRP remains NP-complete in the strong sense if each block is stored on exactly two disks.

Theorem 4. $R\big||M_j| = 2, \mathrm{pmtn}^*, \text{set-up}\big|C_{\max} \leq 2$ *is NP-complete in the strong sense.*

Proof (sketch). For the complete proof, we again refer to [2]. The theorem is proven by a reduction from a special variant

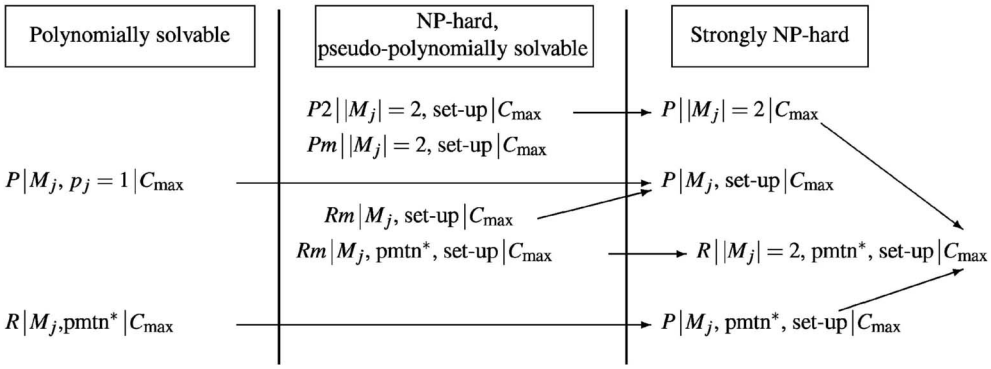

Fig. 2. Complexity diagram of retrieval problems.

of the satisfiability problem in which the number of literals per clause and the number of occurrences per variable is bounded. In the translation, we construct two disks for each variable, corresponding to "true" and "false," and a disk for each clause. For each variable, a block is constructed that has to be retrieved from the "true" disk or the "false" disk. For each occurrence of a variable, a block is constructed that has to be retrieved from the "true" disk or from the clause disk if it is a positive occurrence and from the "false" disk or the clause disk if it is a negative occurrence. The correspondence uses that if an occurrence block is not scheduled on its clause disk, it satisfies the clause. In the translation, each clause disk has space for two occurrence blocks such that each clause is satisfied if all blocks can be scheduled. □

Using the same reduction, we also show in [2] that maximizing the numbe of jobs completed on time 2 is MAX SNP-hard unless $P = NP$. This means that no polynomial time approximation scheme exists for this problem.

It is a well-known result that multiprocessor scheduling problems with preemption but without set-up times can be modeled as a linear programming problem and, consequently, these problems are solvable in polynomial time. As machine eligibility constraints fit in such an LP model [3] and "pmtn^*" is easier than "pmtn," the following corollary, which is added for the sake of completeness, holds.

Corollary 5. *$R|M_j, \mathrm{pmtn}^*|C_{\max}$ is solvable in polynomial time.*

Complexity results for multiprocessor scheduling problems often change if the number of machines is considered to be a part of the problem definition instead of part of the input. Practically, these problems are of interest as they describe the retrieval problems for a given disk array. We start with a complexity analysis of the problems without preemption.

Theorem 6. *$P2||M_j| = 2, \text{set-up}|C_{\max} \leq T$ is NP-complete.*

Proof. The problem is a special case of partition which is known to be NP-complete in the ordinary sense [12]. □

Generalization gives that the following problems are NP-complete as well:

- $Pm||M_j| = 2, \text{set-up}|C_{\max} \leq T$,
- $Pm|M_j, \text{set-up}|C_{\max} \leq T$,
- $Rm|M_j, \text{set-up}|C_{\max} \leq T$.

In case of a stricly positive set-up time, $Rm|M_j, \mathrm{pmtn}^*, \text{set-up}|C_{\max}$ can be proven to be NP-complete in the same way as, in the construction, the schedule is completely filled such that, due to the set-up time, no blocks will be preempted.

The above problems are NP-complete in the ordinary sense. This means that the best we can do for these problems is finding a pseudopolynomial algorithm, which is an algorithm for which the runtime is bounded by a polynomial in the size of the input and the size of the largest number of the input.

Theorem 7. *$Rm|M_j, \text{set-up}|C_{\max}$ is solvable in pseudopolynomial time.*

Proof. The algorithm is a generalization of a dynamic programming algorithm for the knapsack problem [17]. We assign the blocks one by one according to a given block list. We try to solve the question whether or not we can find a schedule that is finished at time T. We use a state definition $(x_1, \ldots, x_m)$ in which m is the number of disks and x_i the amount of transfer time assigned to disk i. We can restrict to values of x_i from $\{0, 1, \ldots, T\}$ such that the number of states equals $(T+1)^m$.

Next, we define F_k as the set of states that can be reached after assigning the first k blocks and start with $F_0 = \{(0, 0, \ldots, 0)\}$. In iteration k, we consider block k of the block list and we can determine F_k with the recurrence relation

$$F_k = \{x + p_{ik}e_i | x \in F_{k-1} \wedge i \in M_k\}, \quad (1)$$

in which e_i is the ith unit vector. We omit the states in which any of the values x_i is larger than T as these states can never lead to a feasible assignment.

In this way, the feasibility question can be reformulated as follows: A feasible assignment exists if and only if $F_n \neq \emptyset$. The complexity of this algorithm is bounded by $O(T^m \cdot n \cdot m)$. As we fixed the number of disks, m is a constant, so the algorithm is pseudopolynomial in the size of the input. □

A similar dynamic programming algorithm can be constructed for the following problems:

- $P2||M_j| = 2, \text{set-up}|C_{\max} \leq T$,
- $Pm||M_j| = 2, \text{set-up}|C_{\max} \leq T$,
- $Pm|M_j, \text{set-up}|C_{\max} \leq T$.

Even in the case where all transfer times and the parameters s and c of the switch time function are rational numbers, this result holds. However, the complexity of the dynamic programming algorithm grows considerably as the number of states depends on the least common multiple of the denominators. This number is polynomially bounded in the largest number of the instance as the number of zones is a constant and the number of different denominators is of the same order of magnitude as the number of zones.

In [2], we prove that $Rm|M_j, \mathrm{pmtn}^*, \text{set-up}|C_{\max}$ can also be solved in pseudopolynomial time.

6 Conclusion

In this paper, we discussed the complexity of retrieval problems for random redundant storage. The block-based retrieval problems can be solved in polynomial time, whereas most time-based retrieval problems are NP-hard, but, in some cases, solvable in pseudopolynomial time. We conclude with an overview of the complexity results in Fig. 2. The arcs in the diagram indicate the relations between the problems. With this diagram, we capture the complexity of the whole range of retrieval problems that result from redundant storage in multimedia servers.

Acknowledgments

The authors thank Jan van Leeuwen and Wil Michiels for their useful comments. Gerhard Woeginger acknowledges support by the START program Y43-MAT of the Austrian Ministry of Science, and Frits Spieksma acknowledges support of EU Grant IST-30027 (APPOL II).

References

[1] J. Aerts, J. Korst, and S. Egner, "Random Duplicate Storage Strategies for Load Balancing in Multimedia Servers," *Information Processing Letters,* vol. 76, nos. 1-2, pp. 51-59, 2000.
[2] J. Aerts, J. Korst, F. Spieksma, W. Verhaegh, and G. Woeginger, "Load Balancing in Disk Arrays: Complexity of Retrieval Problems," Technical Report NL-TN 2002-271, Philips Research Eindhoven, 2002.
[3] J. Aerts, J. Korst, and W. Verhaegh, "Load Balancing for Redundant Storage Strategies: Multiprocessor Scheduling with Machine Eligibility," *J. Scheduling,* vol. 4, no. 5, pp. 245-257, 2001.
[4] J. Aerts, J. Korst, and W. Verhaegh, "Improving Disk Efficiency in Video Servers by Random Redundant Storage," *Proc. Conf. Internet and Multimedia Systems and Applications (IMSA '02),* pp. 354-359, 2002.
[5] P. Berenbrink, A. Brinkmann, and C. Scheideler, "Design of the PRESTO Multimedia Storage Network," *Proc. Int'l Workshop Comm. and Data Management in Large Networks (CDMLarge '99),* 1999.
[6] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking, "Balanced Allocations: The Heavily Loaded Case," *Proc. Symp. Theory of Computing (STOC '00),* pp. 745-754, 2000.
[7] S. Berson, S. Ghandeharizadeh, R.R. Muntz, and X. Ju, "Staggered Striping in Multimedia Information Systems," *Proc. ACM SIGMOD Conf. Management of Data,* pp. 79-90, 1994.
[8] M. Buddhikot and G. Parulkar, "Efficient Data Layout, Scheduling and Playout Control in MARS," *Proc. ACM Multimedia,* pp. 199-212, 1997.
[9] T.S. Chua, J. Li, B.C. Ooi, and K.-L. Tan, "Disk Striping Strategies for Large Video-on-Demand Servers," *Proc. ACM Multimedia,* pp. 297-306, 1996.
[10] E. Dinic, "Algorithm for Solution of a Problem of a Maximal Flow in a Network with Power Estimation," *Soviet Math. Doklady,* vol. 11, pp. 1277-1280, 1970.
[11] G. Gallo, M.D. Grigoriadis, and R.E. Tarjan, "A Fast Parametric Maximum Flow Algorithm and Applications," *SIAM J. Computing,* vol. 18, no. 1, pp. 30-55, 1989.
[12] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco: W.H. Freeman, 1979.
[13] J. Gemmell, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L.A. Rowe, "Multimedia Storage Servers: A Tutorial," *Computer,* vol. 26, no. 5, pp. 40-49, May 1995.
[14] A.V. Goldberg and R.E. Tarjan, "A New Approach to the Maximum-Flow Problem," *J. ACM,* vol. 35, no. 4, pp. 921-940, 1988.
[15] J. Korst, "Random Duplicated Assignment: An Alternative to Striping in Video Servers," *Proc. ACM Multimedia,* pp. 219-226, 1997.
[16] J. Korst, V. Pronk, P. Coumans, G. van Doren, and E. Aarts, "Comparing Disk Scheduling Algorithms for VBR Data Streams," *Computer Comm.,* vol. 21, pp. 1328-1343, 1998.
[17] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations.* New York: John Wiley & Sons, 1990.
[18] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, 1982.
[19] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, 1995.
[20] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *Computer,* vol. 27, no. 3, pp. 17-28, Mar. 1994.
[21] P. Sanders, "Asynchronous Scheduling for Redundant Disk Arrays," *Proc. ACM Symp. Parallel Algorithms and Architectures (SPAA '00),* pp. 98-108, 2000.
[22] P. Sanders, "Reconciling Simplicity and Realism in Parallel Disk Models," *Proc. ACM-SIAM Symp. Discrete Algorithms (SODA '01),* pp. 67-76, 2001.
[23] P. Sanders, S. Egner, and J. Korst, "Fast Concurrent Access to Parallel Disks," *Proc. ACM-SIAM Symp. Discrete Algorithms (SODA '00),* pp. 849-858, 2000.
[24] J.R. Santos and R.R. Muntz, "Peformance Analysis of the RIO Multimedia Storage System with Heterogeneous Disk Configurations," *Proc. ACM Multimedia,* pp. 303-308, 1998.
[25] J.R. Santos, R.R. Muntz, and B. Ribeiro-Neto, "Comparing Random Data Allocation and Data Striping in Multimedia Servers," *Proc. ACM Sigmetrics,* pp. 44-55, 2000.