Improving disk efficiency in video servers by random redundant storage

IMSA 2002
21 mei 2002
final submission

Improving disk efficiency in video servers by random redundant storage

Joep Aerts[1,2], Jan Korst[1], and Wim Verhaegh[1]

[1]Philips Research Laboratories, Prof. Holstlaan 4, WY-21, 5656 AA Eindhoven, The Netherlands
[2]Technische Universiteit Eindhoven, Dept. of Mathematics and Computing Science, Eindhoven, The Netherlands
aertsj@natlab.research.philips.com

ABSTRACT
Random redundant storage strategies have proven to be an interesting solution for the problem of storing data in a video server. Several papers describe how a good load balance is obtained by using the freedom of choice for the data blocks that are stored more than once. We improve on these results by exploiting the multi-zone character of hard disks. In our model of the load balancing problem we incorporate the actual transfer times of the blocks, depending on the zones in which the blocks are stored. We give an MILP model of the load balancing problem which we use to derive a number of good load balancing algorithms. We show that, by using these algorithms, the amount of data that is read from the fast zones is substantially larger than with conventional strategies, such that the disks are used more efficiently.

KEY WORDS
video storage servers, multimedia databases, load balancing, random redundant storage, MILP

1 Introduction

A video server offers video streams to a large number of clients. In the server the video data is stored in blocks on an array of hard disks. The array of hard disks should have sufficient bandwidth to service all clients as well as sufficient storage capacity to store all videos. A continuous video stream is realized by repeatedly reading a data block from the disks. A client sends a request to the server for a certain video. When the client is admitted service, he gets assigned a buffer within the server. The client can consume from this buffer at a variable bit rate and the server has to make sure that the buffers do not underflow or overflow. The buffer strategy that we apply is to request the next data block as soon as the filling of the buffer is below a certain threshold.

An important decision in the design of a video server is the choice of the storage and retrieval strategy that is used in the server. The most common strategies proposed in literature are disk striping strategies. In these strategies the video files are striped over (a subset of) the disks of the disk array [1, 2, 3]. Alternative strategies are based on randomization and replication [4, 5, 6, 7, 8]. In case the bandwidth requirements of the server are the bottleneck instead of the storage requirements, the random redundant data strategies outperform the striping strategies. In this paper we analyze storage strategies based on randomization and replication. In these strategies each block is stored on one or more randomly chosen disks. Whether or not a data block is replicated depends on the storage strategy and, possibly, on the popularity of the movie. When a requested data block is stored more than once, the server has a choice which disk to use for its retrieval. This freedom of choice in the retrieval is used to balance the workload over the disks.

In this paper we assume that the disks are synchronized. The server handles the requests as follows. In each cycle all disks retrieve a batch of blocks and during this cycle the server gathers the incoming block requests from the buffers. As soon as all disks have finished retrieving the assigned block requests of their previous batch, the new set of block requests is distributed over the disks. We want to do this in such a way that the length of a cycle is minimized, which means that the load is balanced over the disks. Consequently, in each cycle we have to solve the following load balancing problem. Given are a set of block requests, and for each request the set of disks on which the corresponding block is stored. The problem is to determine for each requested block which disk(s) to use for its retrieval such that the maximum completion time over all disks is minimized. We describe in this paper an accurate time model, in which we use exact transfer times that depend on the zone in which the blocks are stored. Furthermore, we allow that blocks requests are split up and retrieved from more than one disk, to increase the scheduling freedom.

An advantage of synchronization is that we can use a disk scheduling algorithm such as SCAN [9, 10] to decrease the switch overhead. Furthermore, if blocks are stored twice, once in a fast zone of one disk, and once in a slow zone of another, we can assign the block requests to the disk in such a way that we can read a large percentage of the data from the fast zones. However, regarding the disk efficiency, a disadvantage of synchronization is that it leads to idle time of some of the disks at the end of each cycle. Nevertheless, our simulations show that the fraction of the cycle length that a disk is idle due to synchronization, turns out to be a very small. A final consideration to look at synchronized systems is that the load balancing algorithms can be analyzed mathematically, such that we can give analytical performance bounds, whereas this is much more difficult for asynchronous systems.

The remainder of the paper is organized as follows. In the next section we describe related work in the area of storage and retrieval strategies for video servers. Then, we define the load balancing problem induced by random replicated storage and give a mixed integer linear programming (MILP) model for this problem, in Section 3. In Section 4 we introduce algorithms for the load balancing problem and prove performance bounds. We describe simulation results in Section 5 and end with some concluding remarks in Section 6.

2 Related work

Several papers describe the implementation of multimedia storage servers, such as those describing the PRESTO multimedia storage network [11], the MARS project [12], and the RIO project [13].

Most papers propose disk striping strategies to distribute the video data over the disks. Berson et al. [1], Chua et al. [2], and Santos et al. [8] describe data striping techniques. However, these striping techniques have some disadvantages, especially when used for variable-bit-rate streams. Most striping techniques store 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 and 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 size of the system and that the switch overhead increases, due to the increase in the number of requests [14].

Korst [7] and Santos et al. [8] show that in case of variable bit-rates and less predictable streams, e.g. due to MPEG encoded video or VCR functionality, random replicated storage strategies outperform the striping strategies. In these strategies each block is stored on a randomly chosen disk, and (for some of the blocks) one or more copies are stored on other randomly chosen disk(s). Korst discusses random duplicated assignment and describes algorithms that balance the number of block requests assigned to each disk in a cycle. 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 just 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 multi-zo character of the disks. Sanders [15] describes alternati online scheduling strategies for asynchronous disk contr but also does not consider the disk efficiency opportuniti mentioned above.

Aerts et al. [4] and Sanders et al. [16] prove that ra dom duplicated storage results with high probability in good load balance. Berenbrink et al. [17] give theoretic load balancing results for two online load balancing alg rithms for throwing m balls into n bins, where $m \gg n$. T three papers consider the case of balancing the number requests, but not the actual transfer and switch times.

3 Load balancing

We assume that the disks are synchronized in cycles a that in each cycle a set of blocks has to be retrieved fro the disks. Synchronization means that we cannot start r trieving the next set of blocks before all disks have finish the previous set of blocks. Consequently, the optimizati criterion for assigning the block requests to the disks the completion time of the last disk, i.e. the cycle lengt The completion time of a disk is determined by the sum the transfer times and the switch times. We continue th section with an explanation of the switch time model aft which we can formulate the load balancing problem as MILP problem.

In a cycle each disk gets assigned a batch of (par of) blocks. We assume that the disks use a SCAN-bas sweep-strategy. The total switch time of a batch equals t sum of the individual switch times between the retrievals the blocks of the batch. Each individual switch time co sists of a seek time and a rotational delay. We use the tin of one rotation, r, as an upper bound on the rotational dela For the seek time we use a function that is linear in the nun ber of tracks that the disk head has to cross. For most dis this linear estimation is very accurate, as long as the nun ber of tracks to be passed is not too small. Furthermor we take the worst-case assumption that in each sweep t disk head has to move from the innermost to the outermo track, or vice versa, and that the requests are equally di tributed over the disk [10]. Then, we can compute an upp bound on the total switch time with a function linear in t number of blocks of the sweep. In Section 5 we give pract cal values that we use in the disk model for our simulation For an improved worst-case analysis of the performance a hard disk we refer to [18]. However, for our analysis t simpler model is sufficient.

The transfer time of a block depends on the zone which the block is stored [19]. The outer zones have higher transfer rate than the inner zones, because a disk r tates at a constant angular velocity and outer tracks conta more data. The information of the zone location of bloc on disks is assumed to be known, so the transfer time each block on each disk can be derived. The decision how to distribute the blocks over the zones is defined in t used storage strategy. We re-address this issue in Section when we discuss implementation issues of the simulation.

In the load balancing problem we allow 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 now give a definition of the load balancing problem, that has to be solved in each cycle.

Problem [Load balancing problem (LBP)].
Given are a set J of n block requests that have to be retrieved from a set M of m disks, and for each block request j the set M_j of disks on which the block is stored. Furthermore, the transfer times of the blocks and the parameters 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 maximum completion time of the disks is minimized.

The feasibility variant is defined as the question whether or not an assignment exists that is finished before or at a given time T. □

Now we model LBP as an MILP problem. For each disk i and block j, we introduce a parameter u_{ij} which is 1 if $i \in M_j$ and 0 otherwise. The transfer time to retrieve block j from disk i is denoted by p_{ij}. Furthermore, the total switch time of disk i is approximated from above by $s \cdot n_i + c$, where n_i is the number of blocks assigned to disk i. The switch-slope s and the switch-offset c are given.

For all $j \in J$ and $i \in M$ we introduce a decision variable $x_{ij} \in [0,1]$, indicating the fraction of block j to be retrieved from disk i. Associated with each x_{ij} is a binary variable $y_{ij} = \lceil x_{ij} \rceil$, indicating whether or not block j is (partially) retrieved from disk i. We denote the cycle length, i.e. the completion time of the last disk, by T. Now we can formulate the load balancing problem as an MILP problem as follows.

Minimize T, subject to

$$\begin{array}{ll}
\forall_{i \in M} & \sum_{j \in J} x_{ij} p_{ij} + s \sum_{j \in J} y_{ij} + c \leq T, \\
\forall_{j \in J} & \sum_{i \in M} x_{ij} = 1, \\
\forall_{j \in J, i \in M} & 0 \leq x_{ij} \leq u_{ij}, \\
\forall_{j \in J, i \in M} & y_{ij} \geq x_{ij} \wedge y_{ij} \in \{0,1\}.
\end{array}$$

In [20] we show that the load balancing problem is NP-complete in the strong sense, by a reduction from 3-partition.

4 Algorithms

In this section we describe three algorithms for LBP. The first two are approximation algorithms based on an LP relaxation of the MILP problem. The third one is a list scheduling heuristic. Afterwards we describe a postprocessing step.

4.1 LP relaxation

In the LP relaxation the integrality constraints on the y-variables are dropped. A straightforward way of deriving a solution for LBP is by solving this LP relaxation and rounding up the y-variables. For an optimal solution of the LP relaxation we may assume each y-variable to have the same value as the corresponding x-variable, as $s \geq 0$ and the cycle length has to be minimized. This means that we can omit the y-variables from the formulation. We use an LP solver to solve the resulting LP problem, in which we minimize T subject to

$$\begin{array}{ll}
\forall_{i \in M} & \sum_{j \in J} x_{ij}(p_{ij} + s) + c \leq T, \\
\forall_{j \in J} & \sum_{i \in M} x_{ij} = 1, \\
\forall_{j \in J, i \in M} & 0 \leq x_{ij} \leq u_{ij}.
\end{array}$$

The first approximation algorithm works as follows. We solve the above LP relaxation, round up the y-variables, and compute the value of T of the corresponding MILP problem. This algorithm is called LP rounding. We denote the cost of the solution of LP rounding for an instance I by $S_{\text{round}}(I)$, the cost of an optimal solution of I by $S_{\text{opt}}(I)$, and the cost of the outcome of the LP relaxation by $S_{\text{lp}}(I)$. Then, we can prove the following theorem.

Theorem 1 [Performance bound of LP rounding].
For each instance I of LBP we have

$$\frac{S_{\text{round}}(I)}{S_{\text{opt}}(I)} \leq 1 + \frac{m^2 \cdot s}{n \cdot (p_{\min} + s)}, \qquad (1)$$

in which $p_{\min}$ equals the transfer time of the innermost zone.

Proof. First we give an upper bound on the number of preemptions, as non-integral y-variables cause an increase in the actual cost, compared to the cost of an LP solution. When using the Simplex method [21], the number of non-zero variables in a solution of a linear programming problem equals the number of constraints, which is in this problem $m + n$, where the bounds on the variables are not counted as constraints. As for each $j \in J$ at least one x_{jm} should be larger than 0, the number of preemptions is at most m. This means that $S_{\text{lp}}(I) + m \cdot s$ is an upper bound for the outcome of LP rounding. Furthermore note that $S_{\text{lp}}(I)$ is a lower bound on the optimal cost of instance I and $S_{\text{lp}}(I) \geq \frac{n}{m}(p_{\min} + s)$. With these bounds the stated result can be derived as follows.

$$\begin{array}{rcl}
\frac{S_{\text{round}}(I)}{S_{\text{opt}}(I)} & \leq & \frac{S_{\text{lp}}(I) + m \cdot s}{S_{\text{lp}}(I)} = 1 + \frac{m \cdot s}{S_{\text{lp}}(I)} \\
& \leq & 1 + \frac{m \cdot s}{\frac{n}{m}(p_{\min} + s)} = 1 + \frac{m^2 \cdot s}{n \cdot (p_{\min} + s)}
\end{array}$$

□

In practice, the ratio between n and m depends on the ratio between disk transfer rate and consumption rate of a

number of blocks	50		100		150		200		250	
	avg.	max.	avg.	max.	avg.	max.	avg.	max.	avg.	max.
max-flow	0.262	0.341	0.491	0.559	0.722	0.813	0.951	1.058	1.180	1.302
list sched.	0.244	0.281	0.456	0.492	0.670	0.710	0.885	0.926	1.100	1.140
LP rounding	0.240	0.278	0.435	0.477	0.630	0.664	0.826	0.867	1.021	1.055
LP matching	0.235	0.264	0.429	0.455	0.623	0.647	0.819	0.845	1.014	1.039

Table 1. Average and maximum cycle lengths.

single client, which gives an indication for the number of clients that can be serviced by one disk. For a given set of system parameters this ratio is more or less constant, and consequently, the ratio m^2/n in (1) makes that the performance bound grows in the size of the system. To improve on this, we follow the work of Lenstra, Shmoys, and Tardos [22] to derive a second approximation algorithm. They use LP relaxation to solve a non-preemptive multiprocessor scheduling problem. With a matching description of the LP solution, they prove that a non-preemptive solution can be constructed from the LP solution in which each disk gets assigned at most one of the preempted block requests. In our case this means that the increase in cost using this matching is at most $p_{\max}+s$, where $p_{\max}$ denotes the maximum transfer time. This algorithm is called LP matching and the cost of the solution is given by $S_{\text{match}}(I)$.

Theorem 2 [Performance bound of LP matching].
For each instance I of LBP we have

$$\frac{S_{\text{match}}(I)}{S_{\text{opt}}(I)} \leq 1 + \frac{p_{\max}+s}{\frac{n}{m}(p_{\min}+s)}.$$

Proof. Using the same lower bound as in Theorem 1 and noting that the matching adds at most $p_{\max}+s$ to the LP solution, the stated result can be derived in a similar way as in the proof of Theorem 1. □

4.2 List scheduling algorithm

In Section 5 we compare the two LP based algorithms with a list scheduling algorithm that is based on the linear reselection algorithm (LRS) of Korst [7]. In this algorithm we start with an empty assignment and assign in each step a new block request from the list of blocks to the disk with the smallest resulting time assigned to it. In a second round we reconsider all block requests. We check if a reassignment results in an improvement on the maximum time of the involved disks.

4.3 Postprocessing

The LP matching algorithm and the list scheduling algorithm result in non-preemptive solutions. To improve these solutions we can perform a postprocessing step in which we allow preemption. We try to preempt each block $j \in J$ in such a way that the workload of its disks is more balanced. For duplicate storage we can do this as follows. The fraction x that has to be reassigned form disk i_1 to disk i_2 is
given by $x = \frac{l(i_1)-l(i_2)-s}{p_{i_1j}+p_{i_2j}}$, in which $l(i)$ is the current load
disk i. The order in which the blocks are checked for pr
emption depends on the data placement. In the impleme
tation we start with the blocks for which the transfer tim
are close to each other. The solution after the postproces
ing step is at least as good as the outcome without postpr
cessing, such that the performance bound for LP matchi
remains valid.

5 Simulation

In our simulation each problem instance corresponds to
cycle of the video server. We use random duplicated sto
age and generate 10,000 instances for several numbers
streams, respectively. For each instance we generate
many requests as the number of streams. This correspon
to the situation that in each cycle a block has to be retriev
for each stream, which means that we analyze the syste
for the case that it is fully loaded. We use disk paramete
based on a practical hard disk. We use an array of 10 dis
that have 15 zones. The transfer rate ranges from 45 MB
to 22 MB/s and we use blocks of 1 MB. The parameters
and c of the switch time are 0.0143 s and 0.0093 s, respe
tively. All presented cycle lengths are given in second
The computation time of each algorithm is negligible con
pared to the period length.

We use the following storage strategy for the da
placement on the disks. If one of the two copies is in t
slowest zone, i.e. zone 1 of one disk, the other one is in t
fastest zone, i.e. zone 15 of the other disk. In the disk of o
simulation zone 15 is much larger than zone 1, such that t
copies of the blocks of zone 2 are also in zone 15. Conti
uing this we get a list of possible combinations of zon
for the two copies. Each combination has a probability
occurrence based on the sizes of the zones.

Table 1 presents the average period length and t
maximum observed value for the three algorithms for LB
when 50, 100, 150, 200, and 250 blocks have to be retriev
per cycle. For list scheduling and LP matching we includ
the postprocessing step. For comparison we added the r
sults of a maximum-flow approach [4, 7] in which the tin
information is not taken into account. In that approach t
optimization criterion is the number of blocks assigned
each disk.

The results show that LP matching outperforms t
other two algorithms for LBP. Furthermore, we see a signi
icant decrease in cycle length compared to the convention
max-flow algorithm, especially in the maximum observ

value. Comparing LP matching with the max-flow results, the maximum cycle length decreases on average with 20% and the average cycle length with 13%. We can also see that, except for 50 streams, the average value for max-flow is higher than the maximum observed value for LP matching. Furthermore, note that the maximum observed value for LP matching for 250 streams is even smaller than the maximum observed value for max-flow for 200 streams, so we can admit substantially more streams by using LP matching.

Table 2 illustrates that this load balancing approach leads to a very efficient usage of the disks, meaning that most of the blocks are read from the outer zones. The second column gives for each zone the fraction of the disk capacity. If the zone information is not taken into account, this equals the fraction that would be read from each zone by random storage. The third column gives for LP matching the fraction of blocks read from each zone. In the fourth column the relative increase is presented, which is computed by dividing the value in the third column by the value in the second. The values resulted from a simulation with 150 blocks per cycle.

zone	fract. of disk cap. (i)	LP matching (ii)	rel. incr. (ii/i)
15	0.1411	0.2818	1.997
14	0.1411	0.2724	1.930
13	0.0631	0.1086	1.721
12	0.0751	0.1131	1.506
11	0.0901	0.1024	1.137
10	0.0781	0.0627	0.803
9	0.0360	0.0200	0.555
8	0.0721	0.0234	0.325
7	0.0480	0.0082	0.171
6	0.0450	0.0037	0.082
5	0.0450	0.0024	0.053
4	0.0450	0.0009	0.020
3	0.0330	0.0003	0.009
2	0.0480	0.0002	0.004
1	0.0390	0.0001	0.003

Table 2. Fraction of blocks read in each zone for LP matching algorithm compared to the fraction of the disk capacity.

For the two outermost zones the fraction is approximately twice the fraction that can be expected from the zone sizes. The copies of the blocks stored in the inner zones are hardly used, so they form a sort of back-up. We can conclude that random duplicate storage is a good solution for systems for which the bandwidth capacity of the disks forms the bottleneck instead of the storage capacity, as these simulation results show that the bandwidth is used very efficiently.

A disadvantage of synchronization is that some disks are idle at the end of each cycle. However, the scheduling freedom of duplicate storage combined with the possibility to assign fractions of blocks results in almost perfectly balanced completion times. To illustrate this, we measured the fraction of time that disks are idle due to synchronization. For the LP matching algorithm this fraction equals 0.78%, whereas for the max-flow algorithm this fraction is 5.3%.

6 Concluding remarks

In this paper we described how to exploit preemption and the multi-zone character of hard disks to improve the disk efficiency in video servers. We introduced a random redundant storage strategy in which for each block one of the copies is in a fast zone (the outer half of the disk). The simulations show that this results in a significant increase in the number of blocks that is read from the outermost zones. Furthermore, we showed that the cycle length and its variance decreases significantly for a fixed number of streams.

Former studies [7, 8] have shown that partial duplication already gives enough load balancing freedom. Another way to improve on storage overhead is random striping [6]. Our approach can be implemented for both strategies to improve on disk efficiency. Furthermore, heterogeneous streams can also be embedded in this model.

In this paper we simulated with a fixed number of streams and a fixed block size. For this setting we compared the resulting cycle lengths of several algorithms. A smaller cycle length can be used in the design of a system to improve on optimization criteria like response time or number of clients. The block size is related to the cycle length in such a way that a block must be large enough to provide video during a worst-case cycle. For a certain block size a cycle length is determined for a given failure probability. Given this cycle length a new smaller block size can be determined, and again a cycle length can be computed. In this way we can iteratively converge to a system with minimal block size and cycle length, to improve on response times. Another possibility is to increase the number of streams, instead of decreasing the block size. By using LP matching the number of streams can be increased substantially, as we observed in the previous section. In general, our approach of increasing disk efficiency can be used to improve on general optimization criteria for the design of video servers.

References

[1] S. Berson, S. Ghandeharidazeh, and R.R. Muntz. Staggered striping in multimedia information systems. In *Proceedings of ACM Sigmod Conference 94, International conference on management of data*, pages 79–90, 1994.

[2] T.S. Chua, J. Li, B.C. Ooi, and K.-L. Tan. Disk striping strategies for large video-on-demand servers. In *Proceedings ACM Multimedia '96*, pages 297–306, 1996.

[3] H. Vin, S. Rao, and P. Goyal. Optimizing the placement of multimeida objects on disk arrays. In *Proceedings of the international conference on multimedia computing and systems*, pages 158–165, 1995.

[4] J. Aerts, J. Korst, and S. Egner. Random duplicate storage strategies for load balancing in multimedia servers. *Information Processing Letters*, 76(1-2):51–59, 2000.

[5] J. Aerts, J. Korst, and W. Verhaegh. Load balancing for redundant storage strategies: Multiprocessor scheduling with machine eligibility. *Journal of Scheduling*, 4(5):245–257, 2001.

[6] S. Berson, R.R. Muntz, and W.R. Wong. Randomized data allocation for real-time disk I/O. In *Proceedings of Compcon Conference*, 1996.

[7] J. Korst. Random duplicated assignment: An alternative to striping in video servers. In *Proceedings ACM Multimedia '97*, pages 219–226, 1997.

[8] J. Santos, R. Muntz, and B. Ribeiro-Neto. Comparing random data allocation and data striping in multimedia servers. In *Proceedings of ACM Sigmetrics, conference on measurements and modelling of computer systems*, pages 44–55, 2000.

[9] E. Coffman, L. Klimko, and B. Ryan. Analysis of scanning policies for reducing disk seek times. *SIAM Journal of Computing*, 1(3):269–279, 1972.

[10] Y.-J. Oyang. A tight upper bound of the lumped disk seek time for the scan disk scheduling policy. *Information Processing Letters*, 54(6):355–358, 1995.

[11] P. Berenbrink, A. Brinkmann, and C. Scheideler. Design of the PRESTO multimedia storage network. In *Proceedings of International Workshop on Communication and Data Management in Large Networks (CDMLarge)*, 1999.

[12] M. Buddhikot and G. Parulkar. Efficient data layout, scheduling and playout control in MARS. In *Proceedings of ACM Multimedia Systems*, pages 199–212, 1997.

[13] J. Santos and R. Muntz. Peformance analysis of the rio multimedia storage system with heterogeneous disk configurations. In *Proceedings of ACM multimedia '98*, pages 303–308, 1998.

[14] J. Gemmell, H.M. Vin, D.D. Kandlur, P.V. Rangan, and L.A. Rowe. Multimedia storage servers: A tutorial. *IEEE Computer*, pages 40–49, 1995.

[15] P. Sanders. Asynchronous scheduling for redundant disk arrays. In *Proceedings 12th ACM Symposium on Parallel Algorithms and Architectures*, pages 98–108, 2000.

[16] P. Sanders, S. Egner, and J. Korst. Fast concurrent access to parallel disks. In *Proceedings 11th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2000*, pages 849–858, 2000.

[17] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: The heavily loaded case. In *Proceedings of Symposium on Theory of Computing, STOC*, pages 745–754, 2000.

[18] W. Michiels, J. Korst, and J. Aerts. On the guaranteed throughput of multi-zone disks. submitted to *IEEE Transactions on Computers*.

[19] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27:17–28, 1994.

[20] J. Aerts, J. Korst, W. Verhaegh, and G. Woeginger. Load balancing in disk arrays: Complexity of retrieval problems. Submitted to *IEEE Transactions on Computers*.

[21] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Inc., New Jersey, 1982.

[22] J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–270, 1990.