Efficient, Distributed Data Placement Strategies for Storage Area Networks*

[Extended Abstract]

André Brinkmann †
brinkman@hni.upb.de

Kay Salzwedel ‡
kay@hni.upb.de

Christian Scheideler §
chrsch@upb.de

ABSTRACT

In the last couple of years a dramatic growth of enterprise data storage capacity can be observed. As a result, new strategies have been sought that allow servers and storage being centralized to better manage the explosion of data and the overall cost of ownership. Nowadays, a common approach is to combine storage devices into a dedicated network that is connected to LANs and/or servers. Such networks are usually called *storage area networks* (SAN). A very important aspect for these networks is scalability. If a SAN undergoes changes (for instance, due to insertions or removals of disks), it may be necessary to replace data in order to allow an efficient use of the system. To keep the influence of data replacements on the performance of the SAN small, this should be done as efficiently as possible.

In this paper, we investigate the problem of evenly distributing and efficiently locating data in dynamically changing SANs. We consider two scenarios: (1) all disks have the same capacity, and (2) the capacities of the disks are allowed to be arbitrary. For both scenarios, we present placement strategies capable of locating blocks efficiently and that are able to quickly adjust the data placement to insertions or removals of disks or data blocks. Furthermore, we study how the performance of our placement strategies changes if we allow to waste a certain amount of capacity of the disks.

*Research is partially supported by the DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen.", the DFG/HNI-Graduiertenkolleg "Parallele Rechnernetzwerke in der Produktionstechnik", and the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT)

†Department of Electrical Engineering and Information Technology, and Heinz Nixdorf Institute (HNI). Paderborn University. D-33102 Paderborn, Germany.

‡Graduiertenkolleg. HNI. Paderborn University. D-33102 Paderborn, Germany.

§Department of Mathematics & Computer Science, and HNI. Paderborn University. D-33102 Paderborn, Germany.

1. INTRODUCTION

In the last couple of years a dramatic growth of enterprise data storage capacity (as much as 50-100% annually in most companies) can be observed. A number of factors are contributing to the recent explosion of data that is overwhelming traditional storage architectures. The introduction of enterprise resource planning, on-line transaction processing, e-business, data warehousing, and especially the increasingly media-rich contents found in intranets and the Internet are heavily adding to that load. This causes a tremendous need for scalable storage systems.

It was already known more than 20 years ago that disk arrays can serve as a scalable storage system (of limited character). They are able to provide not only a flexible disk space but also a higher I/O bandwidth when disks are used in parallel. Mechanisms for enabling a higher I/O bandwidth have been pioneered by Patterson *et al.*[14] when they introduced RAID[1]. This technology had a significant impact on the development of storage systems. Today, every major player in the field of storage systems offers RAID arrays. Many of these systems are based on SCSI, which allows only to attach a small number of disks to a processing unit. To manage all its data an enterprise may need more than one such storage system. But the data distribution and placement has to be done locally by each host, which results in a heterogeneous layout. Hence, these systems lack flexibility and suffer from high maintainance costs. As a result, new strategies have been sought that allow servers and storage being centralized and consolidated to better manage the explosion of data and the overall cost of ownership.

Nowadays, a common approach is to combine storage devices into a dedicated network that is connected to LANs and/or servers. Such networks are usually called *storage area networks* (SAN) [19, 9]. Major requirements for these networks are scalability, reliability, availability and serviceability. Availability includes the important demand that SANs can adapt quickly to a changing number of disks. Due to the fact that SANs may consist of a large number of disks, classical data placement strategies like disk striping [7] may cause severe problems. If, for instance, the striping methods defined in the RAID levels 4 or 5 are used as a global place-

[1] RAID means "Redundancy Array of Independent (formerly, Inexpensive) Disks"

ment strategy, any removal or insertion of a disk causes the redistribution of virtually all the data in the system. Current solutions often circumvent this problem by organizing the SAN in clusters of small RAID systems. This, however, has the drawback that complicated data distribution strategies may have to be used to ensure that data requests can be evenly distributed among all sub-systems. (For instance, by splitting popular data sets into many parts or by producing many copies of them that are stored at different sub-systems.) Another approach that is capable of avoiding too many data replacements is the use of spare disks. If, for instance, a disk fails, a spare disk is used to replace it, keeping the number of operating disks fixed. Only once in a while, a number of spare disks may be added to the set of operating disks, which also helps to keep the number of data replacements low. The drawback of using spare disks, however, is that the available I/O bandwidth of the SAN is not fully exploited.

In this paper, we propose new data placement strategies that always keep the data as evenly distributed among the disks as possible (in fact, close to evenly distributed with high probability). They have the ability to adjust quickly to situations in which data or disks are added or removed. In other words, our algorithms allow the disks to be perfectly "assimilated". Hence, spare disks can be completely avoided. Our strategies do not only have the advantage that the location of a block can be computed fast and with very limited space requirements, but that they can also be implemented in a distributed way. These features are of crucial importance to ensure a high scalability, throughput, and adaptability for SANs.

1.1 The Model

For the scope of this paper, we will not consider the problem of routing messages within a SAN. We simply abstract a SAN as a set of n disks, labeled $D_1, \ldots, D_n$, that are completely interconnected. Let $U = \{1, \ldots, p\}$ represent the set of all numbers available for addressing data blocks, where p may be arbitrarily large. All data blocks are assumed to be of equal size. Only a subset of the blocks in U may be defined at a time. We will call these blocks *used blocks* in the following. Any used data block has to be assigned to a unique disk. Note that this does not limit the applicability of our approach, since any uniform striping or duplication technique (for instance, any redundant placement strategy that stores k sub-blocks for each block: $k-1$ resulting from splitting the block into $k-1$ parts of equal size, and 1 resulting from the parity of these sub-blocks) can be interpreted as having a set of equal sized (sub-)blocks that must be assigned to unique disks. Each of the disks has a limited *capacity* C_i, representing the number of data blocks that can be stored on it. The *total capacity* of the system is the sum over all disk capacities and denoted by C_{total}. We will always assume that C_{total} is at least the number of used blocks.

Our aim is to find data placement strategies that ensure that

1. the used data blocks are as evenly distributed among the disks as possible,
2. the data layout is efficiently computable, and
3. adding or removing disks or data blocks results only in a small fraction of redistributions.

By an *even distribution* we mean the minimization of the maximum number of blocks stored on a disk.

To find such placement strategies is a challenging problem if other methods than lookup tables are sought. There are many reasons why lookup tables should be avoided. First of all, lookup tables for large SANs may be huge. Therefore, storing a copy of such a table in each server connected to the SAN may be a tremendous waste of resources. On the other hand, having a central instance in the SAN that stores the lookup table may limit severely the scalability of the SAN, or may lead to an unnecessarily high price (due to the need of expensive, fast hardware for such a center). The alternative approach, storing the lookup table in a distributed way, may cause bottlenecks (e.g. in the case that some component stores information about many popular data blocks) and communication overhead. Furthermore, a breakdown of a single component may lead to a loss of placement information if no fault-tolerant lookup tables are used. Storing a lookup table in the disks of the SAN is also a bad option, since this involves disk accesses, which are magnitudes slower than internal computations or even communication. Instead, we want to find placement strategies that require only a small amount of resources and therefore can be stored and implemented *locally* at every device of the SAN. That disks will have the resources and computational power to support a distributed layout is substantiated by the current efforts of the National Storage Industry Consortium (NSIC). They launched the NASD (Network Attached Storage Devices [`www.nsic.org/nasd`]) project, which is supposed to develop, explore, validate, and document new technologies for network attached storage devices. One of their main premise is that disks have to become smarter and more sophisticated.

We will analyze our data placement strategies from two perspectives: the *layout quality* (which only considers items (1) and (2) above) and the *adaptability* (which considers item (3) above). In the former case, we judge our data layout and give some evidence for its feasibility. The latter case characterizes the behavior in a dynamic setting, where the number of disks or used blocks in the system changes.

1.1.1 Layout Quality

Given some placement strategy, the goal is to show that, for any set $M \subseteq U$ of used data blocks and any number n of disks, items (1) and (2) above hold (in the expected case, with high probability, or with certainty).

1.1.2 Adaptability

Consider any placement strategy S that fulfills item (1). To model the performance of S for the case that disks or data blocks are added or removed, we employ competitive analysis. For any operation that represents the insertion or removal of a disk or a data block, we intend to compare the number of (re-)placements of data blocks performed by S with the number of (re-)placements of data blocks performed by an optimal strategy that ensures that, after every operation, the used data blocks are evenly distributed among the disks. A placement strategy will be called *c-competitive* concerning operation ω if it induces the (re-)placement of at most c times the number of data blocks an optimal strategy would need for ω.

In this work, we only consider the case of anticipated removals of disks and not the case that disks may fail. Such failures require the use of fault tolerant placement strategies, which will be beyond the scope of this paper. Many strategies may be used to allow the reconstruction of lost data. Among them are, e.g. parity layouts [14, 5], declustered layouts [8, 18], or multi-fault tolerant schemes [6]. It is not difficult to extend our strategies so that they not only work for the planned removal but also in case of a failure of a disk.

1.2 Previous Results

The exploration of disk arrays as an efficient and flexible storage system imposes a number of challenging tasks to solve. First of all, one has to find a suitable data layout, i.e. a mapping of blocks to disks, that allows a fast data access (to improve I/O bandwidth one needs to use the disks in parallel). Additionally, the use of disk arrays calls for fault tolerant layouts because such a system is more susceptible to failure due to an increased mean time between failure (MTBF) [4]. Furthermore, the used access strategy (scheduling of requests), space requirements (buffers) and application properties (pattern of requests) have a large impact on the usefulness of such distribution strategies.

The simplest data layout used is *disk striping* [7] which is applied with different granularity in a number of approaches [14, 20, 6, 8, 4]. Here, the data is cut into equal sized blocks and assigned to disks in a round robin fashion so that logically consecutive blocks are put on consecutive disks, cycling repeatedly over all of them. This simple and effective strategy has the disadvantage that the layout is fixed for any number of disks. A change of the array size results in an almost complete redistribution of blocks, which is not feasible.

In [3], Berson *et al.* generalized the idea of disk striping. They proposed a staggered scheme in which consecutive blocks are separated by a certain stride (in disk striping this stride is 0) and compare it to clustered approaches [20] where the disks are partitioned into groups. They claim that their strategy improves the data access when requests with variable bandwidth are concerned. Nevertheless, their approach is still based on unchanging array sizes. When disks fail within a cluster they need to be replaced or the cluster runs in degraded mode and can not tolerate another disk failure.

The application of randomization for data layouts was found to be promising by many researchers [1, 5, 17, 12]. In such a scheme the data blocks are assigned to a random position at a random disk. One of the first who studied random data placement strategies were Mehlhorn and Vishkin [13]. Among other results, they suggest to keep several copies of each data item. In their scheme, if all requests are read requests, the easiest copy will be read. In case of a write request, all copies will be updated. Upfal and Wigderson [21] showed how to obtain a balanced distribution of both read and write requests. They introduced the majority trick: if $2k$ copies are available, accessing $k+1$ of them for both read and write requests always ensures consistent information. Karp, Luby, and Meyer auf der Heide [11] were the first to present precise bounds on the distribution of requests among memory units if several copies are available for each data item. Alemany and Thathachar [1] introduced a randomized data placement for a News on Demand Server. They also apply replication for short term load balancing and showed that such a strategy can get arbitrary close to the maximum server bandwidth.

Birk [5] proposes a similar data layout, but uses parity information instead of replication. This scheme explores redundancy by choosing $k-1$ out of k possible blocks ($k-1$ data and one parity block) depending on the length of the disk queues. The parity information ensures that all blocks can be reconstructed correctly.

The RIO (Randomized I/O) storage server by Santos and Muntz [17, 15] is designed to meet the requirements of real-time systems. They compare the randomized approach with traditional striping techniques [16]. Surprisingly, even in situations for which striping was designed for (regular access pattern), the random allocation method is equally good or better. However, their pseudo-random function is implemented by using a fixed distribution pattern. They derive such a pattern for a data space of size m and apply it repeatedly until the entire address space is covered. This cannot guarantee a correlation free distribution of data blocks.

Recently, it came to our attention that independently an approach similar to ours has been presented to ensure an even distribution of the data blocks among caches (instead of disks in our case) and to ensure that a minimum number of blocks has to be replaced in case of additional or removed caches [10]. The advantage of this method is that the expected time to compute the location of a block is $O(1)$, whereas we require an expected time of $\Omega(\log n)$. However, our strategy needs a much lower space complexity: (disregarding the necessity of $O(n \log n)$ bits to store the locations of the disks/caches in the network and the space for storing a hash function) the computation of the position of a data block requires a data structure of $O(n \log^2 n)$ bits in [10], whereas we only need $O(\log n)$ bits. Furthermore, for the case of n uniform disks, the strategy in [10] is only able to ensure that at most $O(m/n)$ data blocks are stored in any disk, w.h.p., whereas our strategy ensures that at most $(1 + O(\sqrt{(n \log n)/m}))m/n$ data blocks are stored in any disk, w.h.p. To achieve the same degree of balance, the method in [10] would need $\Omega(m)$ bits, which is unacceptably high, since m can be very large.

1.3 New Results

In our approach, we assume that we are given a family H of high quality pseudo-random hash functions mapping the elements in $U = \{1, \ldots, p\}$ to real numbers in the interval $[0, 1]$ ($\log p$-universal hash functions would already suffice for our constructions). The starting point of any of our placement strategies is to choose a hash function h out of H independently and uniformly at random to ensure that used blocks are given values $h(b)$ that are uniformly distributed in $[0, 1]$.

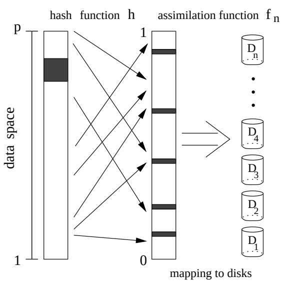

Figure 1: The overall placement strategy.

In order to map the blocks to the disks, we use a so-called *assimilation strategy.* The task of this strategy is to cut the interval $[0,1]$ into a finite set of ranges and assign them to the disks in such a way that

1. the maximal expected number of blocks at a disk is minimized,
2. for any number in $[0,1]$ the hosting disk is efficiently computable,
3. adding or removing disks or data blocks results in moving only a small fraction of $[0,1]$.

Figure 1 illustrates our approach. To ensure that this results in an (close to) even distribution of the used blocks among the disks, we will always assume in the following that m, the number of used blocks, is significantly larger than n, the number of disks. This is obviously a reasonable assumption. It is easy to see that by the use of a pseudo-random hash function we obtain an almost even distribution of data blocks with high probability. Hence, our requirements on data placement strategies formulated in Section 1.1 are fulfilled.

Apart from these properties, our strategy ensures that requests can be close to evenly distributed among the disks (in the expected case) for *any* probability distribution of accessing the used blocks. This is due to the use of pseudo-random hash functions.

We study placement strategies for two scenarios: the case that all disks are *uniform* (i.e., have the same capacity) and the case that disks may be arbitrarily *non-uniform.* The competitive ratios given below hold in the expected case, or multiplied by $(1+o(1))$ also w.h.p.

1.3.1 Uniform disks

For uniform disks, we present a placement strategy that allows to compute the location of any block in $O(\log n)$ steps. Furthermore, it is 1-competitive for adding a disk and 2-competitive for removing a disk.

1.3.2 Non-uniform disks

For non-uniform disks, we choose an approach that separates the capacities of the disks into several levels and uses our strategy for the uniform case in each level. We can show that it is always possible to use less than $O(\log m/\log q)$, where $m=n^q$, levels, even if no two disks are of the same size and the whole capacity is needed to store all blocks.

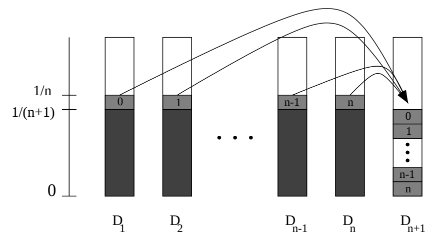

Figure 2: The assimilation of ranges in the uniform case.

Thus, our non-uniform strategy allows to compute the location of a block in $O((\log n \cdot \log m)/\log q)$ steps, with $m = n^q$. Furthermore, we present refined strategies that are $\log m$ -competitive concerning the insertion or removal of data blocks and the insertion or removal of disks. Then, we look at the question by how much the number of levels can be reduced if the balancing condition is relaxed, considering two different cases: a certain percentage of the capacity of each disk is allowed to be wasted, and a certain percentage of the overall capacity is allowed to be wasted. If, for instance, a constant percentage of waste is allowed, then the number of levels reduces to $\Theta(\log n)$ in the first case and may, under certain circumstances, even reduce to a constant in the second case.

2. THE UNIFORM CASE

In this section, we consider the situation that all disks have the same capacity. We present a simple algorithm that was proposed by us in [2]. As will be shown, this algorithm provides a very efficient mapping of the data blocks to the disks.

2.1 A simple placement strategy

As noted in Section 1.3, our placement strategy is based on the assumption that we have a suitable pseudo-random hash function h that maps the used data blocks to real numbers in the interval $[0,1]$. Given a number of n disks, our aim is to cut $[0,1]$ into a finite number of ranges and to assign these ranges in such a way to the disks that the sum of the ranges is equal to $1/n$ for every disk. To simplify the construction, we will denote the set of ranges assigned to disk i simply by $[0,1/n]_i$. Our strategy works as follows.

We start by assigning the interval $[0,1]$ to disk 1. Given n disks, we cut off the range $[1/(n+1),1/n]_i$ from every disk $i \in \{1,\ldots,n\}$ and concatenate these intervals to a range $[0,1/(n+1)]_{n+1}$ for disk $n+1$ by assigning $[1/(n+1),1/n]_i$ to the sub-interval $[(n-i)/(n(n+1)),(n-i+1)/(n(n+1))]_{n+1}$ for every i (see also Figure 2). Given any situation in which there are n working disks, we demand that the interval $[0,1]$ is distributed among these disks as would come out when applying our strategy consecutively from 1 to n disks, no matter what sequence of insertions and removals has been used to arrive at the n disks. This ensures that the distribution of $[0,1]$ among the disks is unique in every situation.

2.2 Analysis of the layout quality

Obviously, our strategy ensures that every disk has the same share of $[0,1]$. This ensures that the used data blocks are

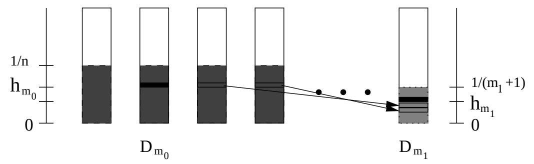

Figure 3: Illustration for the proof of Theorem 1.

very close to evenly distributed among the disks, w.h.p.

In order to compute the location of a block for a given number n of disks, our aim is to follow all replacements of the block from 1 to n disks caused by the movement of ranges described above. The time efficiency of this strategy, of course, crucially depends on the maximum number of replacements necessary for any block. In the following theorem we show that this number is low.

THEOREM 1. *The maximum number of times a block is replaced from 1 to n disks is at most* $\lfloor \log n \rfloor + 2$.

PROOF. For any $i \in \{1, \ldots, n\}$ and any data block b, let $h_i(b) \in [0, 1/i]$ denote the *height* of b at the presence of i disks. Initially, $h_1(b) = h(b)$, and $h_i(b)$ is determined by the position in $[0, 1/i]_j$ of the disk j that contains the range including $h(b)$. Now, consider any block b on disk D_{m_0}. This block has to be moved to disk D_{m_1} when $1/m_1 \leq h_{m_0}(b)$ for the first time. Hence, $m_1 = \lceil 1/h_{m_0}(b) \rceil$. Furthermore, the height of block b on disk D_{m_1} is $h_{m_1}(b) \leq \frac{(m_1 - m_0)}{m_1(m_1-1)}$. The next time b will be moved will be to a disk D_{m_2} with $m_2 \geq \lceil \frac{m_1(m_1-1)}{(m_1-m_0)} \rceil$. Having in mind that $m_1 > m_0$, it is not difficult to show that $\frac{m_1(m_1-1)}{m_0(m_1-m_0)}$ is minimized for $m_1 = 2m_0 + 1$. Hence, for any choice of m_0 and $h_{m_0}(b)$, $m_2 \geq 4m_0 - 2$. From this it follows that the $2i$th position of any block is at a disk with a label at least 4^{i-1}, which proves the theorem. ☐

The theorem implies that $O(\log n)$ calculations suffice to determine the actual position of any block. One may ask, whether the number of replacements of a block can be reduced. The following result shows that our scheme is nearly optimal in this sense.

THEOREM 2. *For any placement scheme, the maximum number of times a block is replaced from 1 to n disks must be at least* $\ln n - 1$.

PROOF. Assume that we have m used blocks. To keep an even distribution of blocks from 1 to n disks, m/i blocks have to be moved from $i-1$ to i disks for every $i \geq 2$. Hence, the total number of replacements is at least $\sum_{i=2}^{n} m/i \geq m(\ln n - 1)$. Thus, the average (and therefore also the worst case) number of times a block is replaced is at least $\ln n - 1$, proving the theorem. ☐

2.3 Analysis of the adaptability

New disks

Adding new disks is easy because we just need to perform the replacements induced by one step of our assimilation technique. Since only an expected fraction of $1/n$ of the blocks is moved when increasing the number of disks from $n-1$ to n, and any placement strategy also has to move a fraction of $1/n$ blocks in this case to keep an even distribution of the blocks, we obtain:

THEOREM 3. *Our assimilation strategy is 1-competitive (in the expected case and* $1+o(1)$*-competitive, w.h.p.) when adding a new disk.*

Removal of disks

Assume that we have n disks, and disk D_i is to be removed. Since all disks have the same capacity, we do this by first reversing our assimilation technique for disk n (which means that all of its blocks are moved back to the other disks). Afterwards, we let disk n take over the role of disk i by simply moving all blocks in disk i to disk n. This results in the following theorem.

THEOREM 4. *Our assimilation strategy is 2-competitive (in the expected case and* $2+o(1)$*-competitive, w.h.p.) when removing a disk.*

Note that our algorithm requires only space for storing the hash function and n. This information can easily be stored locally in the SAN without requiring large resources. Furthermore, this concept allows the disks to react independently on an insertion or removal of disks. All that needs to be done is to announce the new disk or the position of the removed disk to all disks in the system, e.g. via broadcast.

3. THE NON-UNIFORM CASE

We previously assumed a homogeneous set of disks. This seems to be reasonable for a new system. However, if the system needs to be upgraded after a period of time, for instance, to store more data, new disks will have to be added to it. Since disk capacity is changing rapidly, most probably the new disks will have different characteristics than the original ones, such as being faster and having a larger capacity. In this case, assigning each disk the same share of data blocks may lead to a bad performance or may even be impossible. We will restrict ourselves to consider in this section only differences in capacity and not in speed or other characteristics. It is not difficult to see that also other considerations, such as speed, can be incorporated into our algorithms.

Recall, that we aim to minimize the maximum number of data blocks stored in a disk. In this case, some disks may need to use all their capacity to store blocks. Such disks will be called *saturated.*

Our basic approach is to reduce the problem of placing data in non-uniform disks to the problem of placing data in uniform ones. This allows the application of our strategy presented in the previous section. We achieve the reduction by introducing *levels*, starting with level 0. In each level, we attempt to distribute all the (remaining) data blocks evenly among the disks. All blocks that cannot be stored due to saturated disks will be attempted to be placed in the next higher level. Obviously, the number of levels used can be at most n, since a new level is only introduced when a disk saturates.

For each level j, let n_j denote the number of disks participating in this level, i.e., they were not already saturated in previous levels. Clearly, $n_0 = n$. Furthermore, let B_j be the number of blocks each disk is supposed to store in level j. If m is the number of used blocks, then we have $B_0 = \frac{1}{n_0} \cdot m$ and $B_{j+1} = \frac{n_j - n_{j+1}}{n_{j+1}} \cdot B_j$ for every $j \geq 0$. We define $\epsilon_j \in [0,1]$ to be the rate by which the number of disks decreases from level j to $j+1$, i.e., $\epsilon_j = n_{j+1}/n_j$. Hence, $n_{j+1} = \epsilon_j n_j$ and $B_{j+1} = \frac{1-\epsilon_j}{\epsilon_j} \cdot B_j$.

3.1 A lower bound on the number of levels

We start with presenting a lower bound for the maximum number of levels needed by any algorithm that uses our approach.

THEOREM 5. *For any n and $m = n^q$ with $q \geq 2$ there is a collection of n disks for which our approach requires at least $(\log m)/(2 \log q)$ levels for placing m data blocks.*

PROOF. To derive a counterexample, we assume ϵ_j to be equal to some fixed ϵ for all levels. Using the notation above, it follows for level i that $n_i = \epsilon^i n_0$ and $B_i = \left(\frac{1-\epsilon}{\epsilon}\right)^i B_0$. Clearly, for all levels we must have $n_i \geq 1$ and $B_i \geq 1$. We will show that for $\epsilon = 2^{-\frac{1}{k}}$ with $k = \frac{q}{2 \log q}$ these inequalities hold for at least $k \cdot \log n$ levels, which would prove the theorem.

With the choice of ϵ above, we obtain that $n_i = 2^{-i/k} n_0$. This is at least 1 for all i with $i \geq k \cdot \log n$. It remains to show that $B_i \geq 1$ for all $i \geq k \cdot \log n$. We have $\frac{1-2^{-1/k}}{2^{-1/k}} = 2^{1/k}(1 - 2^{-1/k}) = 2^{1/k} - 1$ and

$$2^{1/k} - 1 \geq \frac{1}{2k} \quad \Leftrightarrow \quad 4 \geq \left(1 + \frac{1}{2k}\right)^{2k},$$

which is true for all $k \geq 1$. Moreover,

$$\left(\frac{1}{2k}\right)^{k \cdot \log n} = 2^{-k \cdot \log(2k) \log n} \geq 2^{-(\log m)/2} = \frac{1}{\sqrt{m}}.$$

Thus, $B_{k \log n} = \left(\frac{1-\epsilon}{\epsilon}\right)^{k \log n} \frac{m}{n} \geq \frac{\sqrt{m}}{n} \geq 1$ for all $q \geq 2$, which completes the proof. □

3.2 A naive placement strategy

In this section we propose a first, naive strategy for distributing data blocks among non-uniform disks. The strategy simply works by using a new, independently chosen hash function and applying the strategy presented for the uniform case at each level. The reason for choosing independent hash functions is to avoid that the placements for different levels are correlated. There are simple counterexamples that show that these correlations can result in a very uneven distribution of the data blocks.

For each level j, we apply the uniform strategy in the following way: First, we distribute the interval $[0,1]$ among the n_j disks in the same way as done in Section 2. Assuming that B_j used blocks have to be placed on each disk and disk i has a (remaining) capacity of c_i, we define the *height* of disk i by $\beta_i = \min\{1/n_j, c_i/(B_j \cdot n_j)\}$. All blocks in the interval $[0, \beta_i]_i$ will be placed at disk i and all others will be moved to the next higher level. Note that changing the number of disks or the number of used blocks changes the heights of the disks and therefore may result in replacements of data. In the following, we analyze how bad this can be.

Analysis of the layout quality

The time complexity of locating a block in the system depends not only on the number of replacements within each level but also on the number of levels possible. Hence, it is of crucial importance for an efficient strategy to keep the number of levels small. We will show the following worst case bound.

THEOREM 6. *For any n and $m = n^q$ with $q \geq 1$, the maximum number of levels caused by distributing m blocks among n disks is at most $\frac{3(q+1)}{\log(q+1)} \cdot \log n$.*

PROOF. Using our notation, we derive two equations for the number of blocks in any level i and the number of disks participating in that level: $n_i = \left(\prod_{j=0}^{i-1} \epsilon_j\right) \cdot n$ and $B_i n_i = \left(\prod_{j=0}^{i-1} (1 - \epsilon_j)\right) m$. This implies the following two conditions if i levels are used:

$$\prod_{j=0}^{i-1} \epsilon_j \geq \frac{1}{n} \quad \text{and} \quad \prod_{j=0}^{i-1} (1 - \epsilon_j) \geq \frac{1}{m}.$$

We will show that this can be fulfilled for at most $\frac{3(q+1)}{\log(q+1)} \cdot \log n$ levels.

Suppose, the conditions can be fulfilled for $\ell \geq 3k \log n$ levels, where $k = \frac{q+1}{\log(q+1)}$. Then there must be at least $k \log n$ levels i with $\epsilon_i < 2^{-1/k}$, or at least $2k \log n$ levels with $\epsilon_i \geq 2^{-1/k}$. If $\epsilon_i < 2^{-1/k}$ for at least $k \log n$ levels, then $\prod_{i=0}^{\ell-1} \epsilon_i < 2^{-\log n} \leq 1/n$, which violates the first condition. If $\epsilon_i \geq 2^{-1/k}$ for at least $2k \log n$ levels, then

$$\begin{aligned} \prod_{i=0}^{\ell-1} (1 - \epsilon_i) &\leq (1 - 2^{-1/k})^{2k \log n} \qquad (1) \\ &< \left(\frac{1}{k}\right)^{2k \log n} \leq 2^{-q \log n} = \frac{1}{m}. \end{aligned}$$

Inequality (1) holds, since $(1 - 1/k)^k < 1/2$ for all $k \geq 1$. □

This demonstrates that the number of levels used by our strategy is very close to the lower bound. Theorems 1 and 6 imply the following corollary.

COROLLARY 1. *The number of computations needed to locate any block b is bounded by $O(\log n \cdot \log m / \log q)$.*

Keeping the used data blocks under any circumstances distributed among the disks as prescribed by the naive placement strategy has the disadvantage that it may involve moving many data blocks. For example, the insertion or removal of a single data block may cause a disk to enter or leave a level. This might require to move at least B_j blocks. Thus, any one of these operations can be very expensive, leading to a bad competitive ratio. Hence, we would like to improve our strategy.

3.3 A refined placement strategy

In this section we will propose a refined version of the naive placement strategy that ensures that for any operation we obtain a low competitive ratio under any circumstances. The idea of our refined strategy is to be more lazy in replacing data blocks to ensure an even distribution among the disks.

One possible solution is to modify our naive strategy in a way that, for the case of increasing the number of disks from n to $n+1$, ranges are not simply concatenated when moved from disks 1 to n to disk $n+1$ within a level, but that we use a new pseudo-random hash function to provide new heights in $[0, 1/(n+1)]$ for the blocks in the corresponding ranges. The hash function ensures that for any height β_{i+1} of disk $i+1$, every disk $j \leq i$ has the same number of blocks with new heights at most β_{i+1}. Thus, if we only replace those blocks of new height at most β_{i+1} and keep all other blocks where they are, then we obtain a distribution of the blocks that is (very close to) as even as possible. However, we *treat* the untouched blocks for the next levels as if they would participate there (as they would in the naive strategy). It can be shown that using a new pseudo-random hash function for each replacement ensures the properties that the blocks are evenly distributed among the disks and that every block is replaced at most $O(\log n)$ times in each level, w.h.p. Hence, the bounds shown for the naive strategy also hold for the refined strategy.

The drawback of our new placement strategy is that the location of a block might be ambiguous. It may have been kept back at some level, since some new disk was not able to store it. Also, it could have been moved to the next higher level due to the insertion of blocks. However, in each level, there is only one possible location for any block: the last disk that was able due to its capacity to store it. Hence, for l levels, our new strategy only causes at most l requests to be sent to all possible locations of a block. Since every block is only stored at one location, only one of these requests will involve a disk access and the transport of data along communication links.

We note that the drawback of sending l requests can be easily avoided in many cases: By default, only one request will be sent, namely to the disk at which the corresponding data block is supposed to be when applying the naive placement strategy. Only if the block is not found there, l requests will be sent. Upon accessing the block, it will be moved to its correct position.

3.4 Analysis of the adaptability

Adding and removing data blocks

Adding or removing a block does induce some overhead because the internal structure of our construction needs to be preserved. Suppose, a new block is introduced. Obviously, every placement strategy has to perform the (re-)placement of at least one block (namely, the new one) in this case. In our strategy, this operation can be more complicated. In the worst case, our strategy wants to put a block on a saturated disk D_s. Suppose, the height of that block is less than the height of D_s (otherwise, the block will be moved to the next level and no additional placement will occur). Disk D_s must store that block because we have to keep ranges compact. This may cause another block to be removed from D_s (naturally, the one with the largest height) and moved to the next higher level. The worst case occurs when this situation repeats itself in each level. It can be shown by a similar argumentation that removing a block results in the same number of replacements. Thus, we can conclude the following lemma.

THEOREM 7. *Assuming that the number of introduced levels is ℓ, the refined placement strategy is ℓ-competitive for the removal and $\ell+1$-competitive for the insertion of a block.*

New disks

Suppose that a new disk D_{n+1} of capacity C_{n+1} is added to the system. In order to keep the distribution of blocks among the disks as balanced as possible, any strategy has to replace a certain amount of blocks. Then, two cases have to be considered. If the capacity of the new disk is less than $1/(n+1)$ of all the blocks in the system, then an optimal algorithm replaces just the number of blocks that suffice to saturate this disk. Otherwise, the new disk can store its share completely. In this case, an optimal algorithm will move at least $1/(n+1)$ of all the blocks (but at most C_{n+1}) to this disk.

Surely, the refined strategy will move in both cases the same number of blocks to the new disk. Additionally, however, it will also move other blocks since some previously saturated disks may be able to store additional blocks. This can be viewed as removing blocks from the system. Thus, we can use Theorem 7 to obtain the following result.

THEOREM 8. *Assuming that the number of introduced levels is ℓ, the refined placement strategy is ℓ-competitive when a new disk is added to the system.*

Removal of disks

Our strategy, called *zero-height-strategy*, works as follows: In order to remove a disk, we set its disk height to zero in all levels in which it participates. This ensures that all of its blocks will be moved to the next higher level in the usual way. When a new disk is added to the system, it will take over the positions of the removed drive in as many levels as its capacity allows.

Suppose, D_f denotes a removed disk that stored m_f blocks. Further, assume the last level in which D_f participated is k. The number of replacements performed by an optimal algorithm is at least m_f because it has to move the blocks in D_f to the other disks. The zero-height-strategy achieves the following result.

THEOREM 9. *The zero-height-strategy is $O(\log m/\log q)$-competitive when a disk is removed.*

PROOF. As stated above, an optimal algorithm needs C_f redistributions. The zero-height-strategy has to move C_f data blocks to the remaining disks. This can be viewed as adding new data blocks to the system. Thus, Theorem 7 finishes the proof. □

Hence, incorporating the zero-height-strategy in our refined strategy yields a placement strategy with a low competitive ratio for any operation involving the insertion or removal of disks or blocks. A slight drawback of using the zero-height-strategy is that the number of disks used for the placement may not be equal to the number of working drives. This may lead to a slightly increased computation time as the upper bounds on the number of replacements and levels now depend on maximal number of disks that has ever been in the system (see Theorems 1 and 6).

One may ask whether it is possible to avoid two drawbacks of our construction: 1) that we cannot ensure a unique position of a data block at any time, and 2) that a data block is not guaranteed any more (only w.h.p.) to be replaced at most $O(\log n)$ times within a level of n disks. However, using the approach of defining the unique position of a data block by the naive placement strategy, it is not difficult to see that both properties cannot be avoided together with a low competitive ratio. Thus, a completely new placement approach would be necessary to obtain better results.

4. EXPLOITING SLACKNESS

In many applications, the most important and costly disk feature is not anymore the capacity but the bandwidth of a disk. Assuming capacities of 50 GByte and more, a single disk is able to store up to 15 MPEG-2 movies, but it is not able to serve hundreds of requests demanding the same movie with different starting points. The previously described algorithms are designed to fully exploit the available capacity of the disks. In this section, we investigate the situation that a certain amount of the capacity of the disks is allowed to be wasted. Under that assumption, of course, an even distribution cannot be guaranteed.

Suppose, we would like to explore only a fixed amount of the available capacity. Then, two different approaches can be investigated:

1. Up to a fixed ratio $w \in (0,1)$ of the capacity of each disk is allowed to be wasted. We call this strategy *local slackness strategy*, because it can be checked locally at each disk whether the capacity constraint is fulfilled.
2. Up to a fixed ratio $w \in (0,1)$ of the overall capacity of the system is allowed to be wasted. This strategy is called *global slackness strategy*. It has the advantage that the capacity constraints at the disks can be handled with much more flexibility. However, changes in the number of disks may affect the whole system.

In particular, we are interested in the question whether sacrificing a fixed part of the disk capacity reduces the number of levels and therefore improves the computational complexity for locating a data block.

4.1 The local slackness strategy

There is a major disadvantage when the disk capacity is not allowed to be wasted: disks may oscillate between two levels and consequently, many blocks need to be moved between consecutive levels. To avoid this, we use the following local slackness strategy. Assume, it is allowed to waste a fixed fraction $w \in (0,1)$ of the capacity at each disk. Let $x = w/(1-w)$. Of course, all disks participate in level 0. For each level i with $i \geq 1$, we allow a disk to participate in i only if its remaining capacity is at least x times the capacity used for the previous levels. This ensures that at most a fraction of w of its capacity is dissipated. It is evident that level i is the last level, if $B_i \leq x \cdot \sum_{j=0}^{i-1} B_j$. For all disks participating in a level, the previously described strategies may be used.

An important consequence of our participation rule is that as soon as the capacity threshold to take part in a level is above the capacity necessary to store an even share of the blocks, there will be no higher level. Moreover, oscillations of disks between two levels and the amount of blocks moved to higher levels can be reduced significantly.

It is easy to check that, as long as the number of used blocks is at most $(1-w) \cdot C_{total}$, our placement strategy will always be able to place all blocks. Furthermore, the maximum relative deviation from an even distribution of the blocks among the disks can be at most w.

4.1.1 An upper bound on the number of levels

In the following, we present an upper bound for the maximum number of levels caused by the local slackness strategy.

THEOREM 10. *For any w, m and n, the maximum number of levels caused by the local slackness strategy is at most $\log_{1+x} n + 1$, where $x = w/(1-w)$.*

PROOF. For every level i, let $\delta_i = 1/\epsilon_i$. It can be shown that the number of blocks stored at a disk in level $i+1$ is at most $B_{i+1} = (\delta_i - 1)(B_i - x\sum_{j=0}^{i-1} B_j)$. If level $i+1$ is not the last level, then we have $B_{i+1} \geq x \sum_{j=0}^{i} B_j$. It follows that $(\delta_i - 1)(B_i - x\sum_{j=0}^{i-1} B_j) \geq x\sum_{j=0}^{i} B_j$ and $(\delta_i - 1)B_i \geq xB_i$. In this case, $\delta_i - 1 \geq x$ and therefore $\delta_i \geq 1 + x$, yielding the theorem. □

Note that the number of levels only depends on n and x, but *not* on m as it was the case in Section 3. This demonstrates that, for our approach, sacrificing disk space helps to improve the computational complexity for block location.

4.1.2 A lower bound on the number of levels

In this section, we present a lower bound on the maximum number of levels showing that our upper bound above is close to optimal.

THEOREM 11. *For any n and any w, the maximum possible number of levels is at least $\log_{4(1+x)^2} n + 1$.*

PROOF. To derive a counterexample, we assume ϵ_i to be equal to some fixed ϵ for all levels and that a fixed ratio ξ of

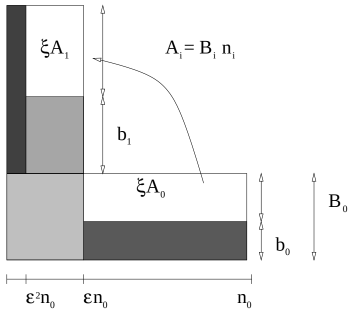

Figure 4: Illustration for the proof of theorem 11.

the stored data of level i is transferred to level $i+1$ (see figure 4). It follows for level i that $n_i = \epsilon^i n_0$ and $B_i = \left(\frac{\xi}{\epsilon}\right)^i B_0$. Furthermore we introduce $k = \frac{1-\epsilon-\xi}{1-\epsilon}$ and $b_i = k \cdot B_i$. For all levels we must have $n_i \geq 1$ and $b_i \geq x \cdot \sum_{j=0}^{i-1} B_j$. In the following we will show that the second condition could be fulfilled for any i.

$$\begin{array}{rrcl}
 & b_i & \geq & x \cdot \sum_{j=0}^{i-1} B_j \\
\Rightarrow & k \cdot \left(\frac{\xi}{\epsilon}\right)^i \cdot B_0 & \geq & x \cdot \sum_{j=0}^{i-1} \left(\frac{\xi}{\epsilon}\right)^j \cdot B_0 \\
\Rightarrow & k \cdot \left(\frac{\xi}{\epsilon}\right)^i & \geq & x \cdot \frac{\left(\frac{\xi}{\epsilon}\right)^i - 1}{\frac{\xi}{\epsilon} - 1} \\
\Rightarrow & k \cdot \left(\frac{\xi}{\epsilon}\right)^{i+1} - (k+x) \cdot \left(\frac{\xi}{\epsilon}\right)^i + x & \geq & 0 \\
\Rightarrow & k \cdot \left(\frac{\xi}{\epsilon}\right)^{i+1} - (k+x) \cdot \left(\frac{\xi}{\epsilon}\right)^i & \geq & 0 \\
\Rightarrow & k \cdot \left(\frac{\xi}{\epsilon}\right) - (k+x) & \geq & 0
\end{array}$$

We now choose $\epsilon = \frac{1}{4(1+x)^2}$ and $\xi = \frac{1}{2(1+x)}$. Then $k \cdot \left(\frac{\xi}{\epsilon}\right) - (k+x) \geq 0$ for any $x \geq 0$ and any $w \geq 0$, which proves the theorem. ☐

4.2 The global slackness strategy

Next, we consider the problem of finding a way of organizing the n disks into g groups of disks of equal capacity so that at most a fraction of w of the overall capacity of the system is wasted. Clearly, if there are at most g different capacities then our placement strategies create at most g many levels. To prove bounds on g, we introduce some notation. Recall that for every disk i, C_i denotes its capacity. W.l.o.g. we assume that the disks are ordered such that $C_i \geq C_{i+1}$ for all $i \geq 1$. We define the *relative height* of disk i by $\gamma_i = C_i/C_1$ and the *area* covered by the disks by $\Gamma = \frac{1}{n}\sum_{i=1}^{n} \gamma_i$. Obviously, $1/n \leq \Gamma \leq 1$.

4.2.1 An lower bound on the number of levels

In this section we prove the following theorem.

THEOREM 12. *For any Γ and w, there exists a collection of disks such that $g \geq (1-w)\ln(1/\Gamma)$.*

PROOF. Suppose that it is possible to organize the disks into g capacity groups so that at most $w\Gamma$ of the capacity is wasted. Then there must be g integers $1 \leq i_1 < i_2 < \ldots < i_g \leq n$ (and additionally $i_0 = 0$ and $i_{g+1} = n$) with the property that if, for all $j \in \{1, \ldots, g+1\}$, all disks with numbers in $\{i_{j-1}+1, \ldots, i_j\}$ is given a height of γ_{i_j} (resp. a capacity of $\gamma_{i_j} C_1$), then the total capacity of the g groups is at least $(1-w)\Gamma$. For every $i \in \{1, \ldots, n\}$, define $A_i = \frac{i}{n} \cdot \gamma_i$. Then, $\sum_{j=1}^{g} A_{i_j} > (1-w)\Gamma$. This implies that there must exist a j with $A_{i_j} > (1-w)\Gamma/g$. We will show up to which g it is possible to avoid this, which implies that this g is a lower bound for the worst case number of groups needed.

Our aim is to find the maximum g for which it is possible to set $A_i \leq (1-w)\Gamma/g$ for all $i \in \{1, \ldots, n\}$. Since $A_i = \frac{i}{n} \cdot \gamma_i$, it follows that $\gamma_i \leq n(1-w)\Gamma/(i \cdot g)$. Since $\gamma_i \leq 1$ for all i, the maximum possible area that can be covered under this restriction is

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n} \min\left[1, \ \frac{n(1-w)\Gamma}{i \cdot g}\right] \\
\geq\ & \frac{1}{n}\left(\frac{n(1-w)\Gamma}{g} + \sum_{i=n(1-w)\Gamma/g+1}^{n} \frac{n(1-w)\Gamma}{i \cdot g}\right) \\
\geq\ & \frac{1}{n}\left(\frac{n(1-w)\Gamma}{g} + (\ln n - \ln(n(1-w)\Gamma/g))\frac{n(1-w)\Gamma}{g}\right) \\
\geq\ & \frac{(1-w)\Gamma}{g} + \ln\left(\frac{g}{(1-w)\Gamma}\right) \cdot \frac{(1-w)\Gamma}{g}
\end{aligned}$$

This is at least Γ if, for $\epsilon = (1-w)/g$,

$$\epsilon\,(1 + \ln(1/(\epsilon\Gamma))) \geq 1\ ,$$

which is equivalent to $\Gamma \leq \frac{1}{\epsilon} \cdot \mathrm{e}^{1-1/\epsilon}$. This is the case if $\epsilon \geq 1/\ln(1/\Gamma)$, that is, $g \leq (1-w)\ln(1/\Gamma)$, which proves the theorem. ☐

The theorem shows that, somewhat surprisingly, the number of groups required to cover a certain percentage of Γ is *not* independent of Γ. Most definitely, our lower bound is not sharp. We assume that the correct bound may be close to $g \geq \frac{\ln(1/\Gamma)}{w}$, but a proof for this is not known yet.

4.2.2 An upper bound on the number of levels

Finally, we present an upper bound for g that is close to the lower bound if Γ and w are large.

THEOREM 13. *For any Γ and w, $g \leq 2\log^2(1/\Gamma)/w$.*

PROOF. We will construct a staircase with g stairs by an induction argument. Our starting point is the point $(x_0, y_0) = (0, 1)$, where x represents the disk number divided by n and y represents the height. Assuming that, for some $i \geq 0$, we are currently at a point (x_i, y_i) with $x_i \geq \sum_{j=1}^{i} 2^j\Gamma$ (which implies that $y_i \leq 1/2^i$), we will show that we need at most $2\log(1/\Gamma)/w$ stairs to get to a point (x_{i+1}, y_{i+1}) with $x_{i+1} \geq \sum_{j=1}^{i+1} 2^j\Gamma$. If this is true, then we require at most $2\log^2(1/\Gamma)/w$ stairs to reach a point (x, y) with $x \geq 1$, which ends the construction and proves the theorem.

For $i = 0$, our assumption for the induction step clearly holds. So assume in the following that the assumption holds for some $i \geq 0$. Then we show that it also holds for $i+1$.

Suppose as a worst case that we start at the point $(x_i, y_i) = (\sum_{j=1}^{i} 2^j\Gamma, 1/2^i)$. In the following, let the x-coordinates be defined relative to x_i (that is, instead of x' we use $x' - x_i$). Consider some fixed positive number A. We intend to bound the number of stairs of waste at most A, each, it takes to reach a point (x'_{i+1}, y'_{i+1}) with $x'_{i+1} \geq 2^{i+1}\Gamma$ (relative to x_i). Clearly, such a point is reached if we end up at a point (x, y) with the property that either $x \geq 2^{i+1}\Gamma$ or $y \leq 0$. Assume that we need m stairs of wastage A for this, numbered from 1 to m. Suppose that stair s represents a rectangle of height γ_s and width b_s. Then, $\gamma_s \cdot b_s \geq A$. As a worst case, we assume that $\gamma_s \cdot b_s = A$. We require that $\sum_{s=1}^{m} \gamma_s \leq 2^{-i}$ and $\sum_{s=1}^{m} b_s \leq 2^{i+1}\Gamma$. In order to find the maximum m for which these restrictions can be fulfilled, we assume that $\gamma_s = \epsilon A$ for all s, where $\epsilon \in [0, 1]$. In this case, $m \cdot \epsilon A \leq 2^{-i}$ and $m \cdot 1/\epsilon \leq 2^{i+1}\Gamma$. That is, $m \leq 1/(2^i\epsilon A)$ and $m \leq 2^{i+1}\epsilon\Gamma$. To maximize m, we choose ϵ such that $1/(2^i\epsilon A) = 2^{i+1}\epsilon\Gamma$. This is the case for $\epsilon = 1/(2^i\sqrt{2A\Gamma})$. From this we obtain $m = \sqrt{2\Gamma/A}$. The wasted area, W, is therefore limited to $m \cdot A = \sqrt{2\Gamma A}$. We require that $W \leq w\Gamma/\log(1/\Gamma)$ to end up with a total wasted area of at most $w\Gamma$. In this case,

$$\sqrt{2\Gamma A} \leq \frac{w\Gamma}{\ln(1/\Gamma)} \quad \Rightarrow \quad \sqrt{A} \leq \frac{w\sqrt{\Gamma}}{\sqrt{2}\log(1/\Gamma)} \; .$$

Plugging this in the equation for m, we obtain

$$m \leq \sqrt{2\Gamma} \cdot \frac{\sqrt{2}\log(1/\Gamma)}{w\sqrt{\Gamma}} = \frac{2\log(1/\Gamma)}{w} \; .$$

Summing this over all $\log(1/\Gamma)$ induction steps yields the theorem. $\square$

The theorem shows that if Γ is large, then the number of levels may be much lower than $\log n$. No other approach considered above was able to achieve this. Hence, it may be interesting to look at efficient solutions in this direction.

5. ACKNOWLEDGMENTS

We would like to thank the anonymous referee for the very helpful comments.

6. REFERENCES

[1] J. Alemany and J. S. Thathachar. Random striping news on demand servers. Technical report, University of Washington, Department of Computer Science and Engineering, February 1997.

[2] P. Berenbrink, A. Brinkmann, and C. Scheideler. Design of the PRESTO multimedia data storage network. In *Proceedings of the Workshop on Communication and Data Management in Large Networks (INFORMATIK 99)*, October 1999.

[3] S. Berson, S. Ghandeharizadeh, R. R. Muntz, and Xiangyu Ju. Staggered striping in multimedia information systems. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 79–90, 1994. *SIGMOD Record* 23(2), June 1994.

[4] S. Berson, L. Golubchik, and R. R. Muntz. Fault tolerant design of multimedia servers. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 24(2):364–375, June 1995.

[5] Y. Birk. Random RAIDs with selective exploitation of redundancy for high performance video servers. In *Proceedings of 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, April 1997.

[6] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: an optimal scheme for tolerating double disk failures in RAID architectures. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 245–254. IEEE Computer Society TCCA and ACM SIGARCH, April 18–21, 1994.

[7] A. L. Chervenak, D. A. Patterson, and R. H. Katz. Choosing the best storage system video service. In *The Third ACM International Multimedia Conference and Exhibition (MULTIMEDIA '95)*, pages 109–120, New York, November 1996. ACM Press.

[8] Mark Holland and Garth Gibson. Parity declustering for continuous operation in redundant disk arrays. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 23–35, 1992.

[9] Adaptec Inc. Fibre channel, storage area networks, and disk array systems – a white paper, 1998.

[10] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 654–663, 4–6 May 1997.

[11] R. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 318–326, 1992.

[12] J. Korst. Random duplicated assignment: An alternative to striping in video servers. In *Proceedings of The Fifth ACM International Multimedia Conference*, pages 219–226, November 1998.

[13] K. Mehlhorn and U. Vishkin. Randomized and deterministic simulation of PRAMs by parallel machines with restricted granularity of parallel memories. In *9th Workshop on Graph Theoretic Concepts in Computer Science*. University of Osnabrück, Germany, 1983.

[14] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, pages 109–116, June 1988.

[15] J. R. Santos and R. Muntz. Performance analysis of the RIO multimedia storage system with heterogeneous disk configuration. In *ACM Multimedia 98*, pages 303–308, September 1998.

[16] J. R. Santos and R. R. Muntz. Comparing random data allocation and data striping in multimedia servers. Technical report, University of California, Los Angeles, Computer Science Department, November 1998.

[17] J. R. Santos, R. R. Muntz, and S. Berson. A parallel disk storage system for realtime multimedia applications. *International Journal of Intelligent Systems*, 13(12):1137–1174, December 1998.

[18] E. J. Schwabe and I. M. Sutherland. Flexible usage of redundancy in disk arrays. *MST: Mathematical Systems Theory*, 32, 1999.

[19] D. Tang. Storage area networking – the network behind the server. Technical report, Gadzoox Microsystems, 1997.

[20] F. A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID: A disk array management system for video files. In *Computer Graphics (Multimedia '93 Proceedings)*, pages 393–400. ACM, August 1993.

[21] E. Upfal and A. Wigderson. How to share memory in a distributed system. *Journal of the ACM*, 34(1):116–127, 1987.