Alcom-FT Technical Report Series
ALCOMFT-TR-03-115

Algorithmic Approaches for Storage Networks

Kay A. Salzwedel *

HEINZ NIXDORF INSTITUTE
Paderborn, Germany
nkz@upb.de

1 Introduction

Persistent storage in modern computers is usually realized by magnetic hard disk drives. They form the last, and therefore the slowest level in the memory hierarchy. Disk drive technology is very sophisticated and complex, making an accelerated growth in disk capacity possible. Nowadays, a single of-the-self disk drive is capable of storing up to 180 GB and this number is doubled every 14 – 18 month. Nevertheless, this is not sufficient to manage the ever growing data volumes. The fast evolving processing power of modern computers, the global availability of information, and the increasing use of mixed-media contents demand for more flexible storage systems. The easiest way of providing almost unlimited storage capacity is the use of many disk drives in parallel. Unfortunately, a straightforward solution does not exploit the full potential of storage networks. As an example, suppose there is the need to add further disk drives to an existing system due to changed capacity demands. Without a redistribution of already stored data the capacity problem is solved but the addition of new (and usually faster) disks does not improve the overall performance of data accesses. There might exist very popular data on an 'old' disk resulting in a heavy load for that disk and leading to a bottleneck in the system.

When it comes to very large data sets the best access strategies are given by *External Memory Algorithms*, i.e. organize the data in such a way, that the number of I/O operations is minimized. In general, storage networks are on a different abstraction level and motivated by the growing amount of data, but external memory algorithms usually define a data layout and, hence, are one possibility to manage a number of disk drives in parallel. So, why not always use the data layouts imposed by external memory algorithms? Looking at the previous chapters two drawbacks of external memory algorithms become apparent. First, one has to find efficient and practical algorithms for all the sub-problems given in an application, and that is not always possible (see Chapter **??**). And second, and more important, one has to be better then the underlying operating system. It is one of the key features of common operating systems to handle accesses to the memory hierarchy very efficiently. This is done in a general purpose way by asynchronous I/O operations, virtual memory techniques, or the use of

* Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

buffer caches. Hence, a possible implementation needs to bypass the disk access strategies of the operating system and implements its own external memory behavior. This is a rather complex task and very seldom acceptable. As far as we know, only large data base systems (see Chapter **??**) implement their own data access model.

As an alternative, it is possible to use specialized kernels (like micro kernels or exokernals) which allow the easy implementation of any memory policy. But this approach has the major disadvantage that all the already implemented software has to be adapted to this kernel which might result in a very complex task. Because of this, researchers implementing external memory algorithms stick to widespread, open-source kernels like LINUX.

One might think that there should be libraries to ease the implementation of external memory algorithms. As mentioned in Chapter **??** there exists some support but these libraries suffer from the same problem. They need to provide data access algorithms which might be broken by the access strategy of the operating system.

This all adds up to the need for more general management of a assemblage of disk drives. A storage network is a collection of disk drives which are connected by a network. Each of the disks is characterized by its capacity and its bandwidth (see Fig. 1). The underlying topology or architecture of the network is unrestricted. Hence, we are not concerned with network properties like congestion or latency. It may vary from disks attached to an SCSI bus up to very complex networks like peer-to-peer (P2P) networks where the disks drives are associated with server nodes. In such an overlay network, a number of server work together to solve special tasks like searching. There is no central instance to control the behavior and server might enter or leave the network at any time.

Having a closer look at the above mentioned abstraction, two crucial observations become apparent. First, larger demands for capacity are easily solved by attaching more disks to the network. Nevertheless, the data need to be distributed in an intelligent way to provide an even utilization of disks. Only a good distribution can ensure the scalability in terms of data requests when the number of disk drives is increased. Second, a storage network offers the possibility of parallel data accesses provided that the data resides on different disk drives. This improves not only the access time but also the disk utilization because large data requests are served by more then one disk and, hence, are more balanced.

But using storage networks also imposes an inherent requirement – availability. Every disk has an estimated operation time during which an error is rather unlikely to occur. This entity is called *mean time to/between failure* (MTTF or MTBF, respectively) and for modern drives it is approximately 1,000,000 hours. But for a whole storage network, the MTBF decreases significantly. This is due to the fact, that it depends on the failure rate λ_{sys} [Swa92, Gib91] which is simply the sum of the failure rates λ_i of each participating disk[1].

[1] This is only true under the assumption that the failure rate is constant which is equal to an exponentially distributed time to failure and the failures are independent of each other.

The observations made above can be formalized to the *data distribution problem*. It has to give an answer to the following question: Where should the data be stored so that any data request can be answered as fast as possible? Furthermore, the data distribution has to ensure *scalability*. The more disks are in a storage network the more data requests should be answered or the faster should the data be delivered. This can only be done if the data is evenly distributed over all disks.

In recent years, modern storage systems move towards distributed solutions. Such systems usually consist of common components connected by standardized technology like Ethernet or FibreChannel. Hence, it is rather unlikely that all components have the same characteristic concerning capacity or performance. This leads to the *heterogeneity* problem. Not exploiting the different properties of each component results in a waste of capacity and performance.

As noted above, the data volume is annually growing. This implies a new challenge to storage networks because they should easily *adapt* to changing capacity/bandwidth demands. Allowing the addition of disks would be an easy task, if the space/access balance is ignored. One simply extends with new disks and maps new data to the added disks. This will result in a distribution that cannot increase the access performance for all data even when newer and faster disks are added. The only way to achieve both goals, space balance and the addition of disks, is to redistribute some data. This property is measured by the *adaptivity*. Naturally, the less data is redistributed the more adaptive is the underlying strategy.

The above discussion can be summarized by defining a number of requirements a storage network has to meet.

1. **Space and Access Balance:** To ensure good disk utilization and provide scalability the data should be evenly distributed over all disk drives. The data requests are usually generated by a file system above the storage network. Hence, the storage network has no knowledge about the access pattern and must handle any possible request distribution.
2. **Availability:** Because of the increased sensitivity to disk failures storage networks need to have some redundancy and implement mechanisms to tolerate the loss of data.
3. **Resource Efficiency:** Redundancy implies a necessary waste of resource. Nevertheless, systems should use all its resources in an useful way. Some problems, e.g. adaptivity, are easily solved if large space resources are available. But especially for large networks it is infeasible to provide these resources.
4. **Access Efficiency:** Stored data that cannot be accessed is lost data. Access efficiency defines *time* and *space* requirements needed to access an arbitrary data element. This property is of crucial importance when large networks are concerned.
5. **Heterogeneity:** Heterogeneity describes the capability to handle disks of different size or different bandwidth.
6. **Adaptivity:** The data volume is constantly growing and even generous planned system can quickly run out of space. This property measures the

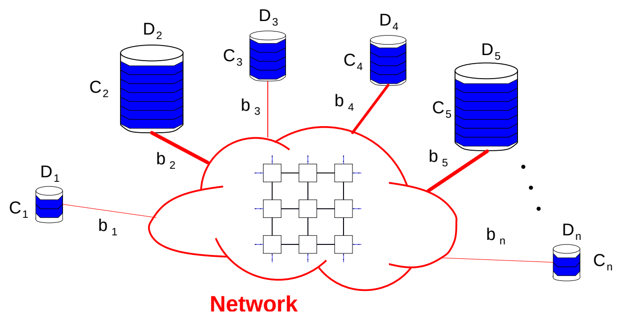

Fig. 1. An arbitrary storage network consists of a number of disks $D_1, \ldots, D_n$ and a connecting network. Each of the disks D_i is characterized by its capacity C_i and its bandwidth b_i.

ability to adapt quickly to a changed situation like the addition of a new disk drive.

7. **Locality:** Locality describes the degree of communication required to answer any data request. One can distinguish centralized and distributed approaches. In this article, only distributed systems are of interest.

It seems impossible to meet all the above requirements optimally, especially heterogeneity and adaptivity are challenging if space/access balance and resource efficiency must be guaranteed. Nevertheless, we will introduce some randomized techniques that come very close to optimal solutions.

1.1 Organization of the Paper

In the next section we define a storage network and introduce all the relevant notations. The remainder of the paper will give an overview of techniques and algorithms used to achieve the above mentioned properties. We will focus on rather new techniques handling heterogeneity and adaptivity because these are the most challenging questions and they will become more important in the future. The last chapter will summarize the paper and give a short conclusion.

2 Model

A *storage network* is a collection of n disks labeled $D_1, \ldots, D_n$, where each disk is characterized by a *capacity* C_i describing the storage capabilities and a *bandwidth* b_i stating the average data transfer time. For simplification reasons, lets assume that the maximal number of disks in one system is bounded by N. The capacity of the whole network can be expressed by $S = \sum_{1 \le i \le n} C_i$. The communication network may be arbitrary as long as all disks are connected. Network specific performance measures like congestion and latency are not considered. A storage network is called *homogeneous* (or *uniform*) if all disks have the same characteristic. Otherwise, the system is *heterogeneous* (or *non-uniform*).

The data to be stored is given by the set $U = \{1, \ldots, p\}$ of *data elements* of equal size. The size of the universe out of which data elements are drawn is denoted by M. In general, we call all operations *data accesses* that read or write data elements. As mentioned in the introductory Chapter **??** the data in each storage hierarchy is accessed in chunks of size B. This modeling is even more accurate when it comes to disk drives because data access involves mechanical components. The time to access a data item is dominated by the time of moving the mechanical read/write head to the right track on the disk. The actual data access is very fast as soon as the head is settled and, furthermore, depends only on rotation speed (modern disk drives spin with 5k up to 15k rotation per minute). In our model, the size of the accessed data chunk B is insignificant. Hence, to avoid confusion, the data on disk is measured in *data units* of equal size (or size B to be precise). For simplification reason, assume w.l.o.g. that the size of a data element equals the size of a data unit. Furthermore, the data elements are ranked by an arbitrary order (e.g. the block address). Any of the p data units may be completely or partly *replicated*. Let $r \in \mathbb{R}$ denote the replication factor by which all data units are extended, e.g. a factor of 1 indicates that every data element has another copy.

The parallel access of data units is modeled by the *stripe* concept. A stripe is a collection of l consecutive data units (consecutive in the defined order of data elements). If the data units are partially replicated, the stripe also contains redundant data units (like parity units). The access to a stripe is fully parallel if all its units reside on different disks.

The number of data units and replication units to be placed in the system is defined by $m = \sum_{1 \leq i \leq p} i + r \cdot i$. A *data distribution strategy* for a storage network defines the replication, the mapping to the n disks, and the access strategy to data units.

For the description of randomized solutions we use the common balls-into-bins model [Ste98]. Here, the data elements are associated with balls and the disks withs bins, respectively.

3 Space and Access Balance

One of the major goals of storage networks is the decrease of overall access time to data elements. In general, this time is strongly correlated to disk utilization if the network behavior is neglected. When a disk gets too many requests the track search time will be increased significantly resulting in disk head threshing in the worst case. Thus, the data elements and the data accesses must be balanced over all participating disks. By the *space balance* of a storage network we mean the property of always mapping even loads to the disks. This will also ensure a complete use of the systems capacity as long as the system is homogeneous. Furthermore, the space balance must guarantee that data units in a stripe are hosted by different disks to ensure parallel access and fault tolerance. The *access* *balance* property describes the capability of distributing the data accesses evenly over the disk drives. It ensures an even disk utilization, if all data elements are equally likely to be accessed. Space and access balance are mutually dependent.

Furthermore, the data access patterns are usually generated by the underlying file system (see Chapter **??**). These patterns can only be predicted if there exists knowledge about the file system itself and the access pattern to each data file. Because this knowledge is hard to predict and may be subject to rapid changes a data distribution cannot take this into account. Hence, a good layout tries only to distribute the data elements as evenly as possible.

Nowadays, disk storage is rather cheap so the question may arise why one is concerned with space balance at all. There are many reasons to grapple with space balance. First of all, the trend of cheaper disks is easily compensated by the annually growing data volume. Furthermore, when the data is evenly balanced over the disk drives the number of accesses served in a given time can be increased because more disks are participating and can be kept busy.

The techniques to achieve space balance are quite simple. They can be grouped into deterministic and randomized approaches. The main deterministic technique is *striping* [Joh84, Kat88, GM86]. Here, the ith data element is put on disk $D_{i \bmod n}$ on position $i \operatorname{div} n$. Giving this simple scheme, full stripes (one data unit from each disk) can always be accessed in parallel. Furthermore, the total capacity can be exploited as long as all disks have the same capacity. There are a number of variations of disk striping. Especially in multimedia systems, it might be of interest not to stripe over the whole array because one wants to achieve a predefined accumulated bandwidth (e.g. the streaming rate of a media object). The natural extension to simple striping would be to group the disks into clusters of stripe length l and use simple striping within the cluster.

But how can the optimal length of a stripe and the size of a data unit be determined for a given demand? In [Pat90] the authors synthesize rules for deriving these parameters. They could show that the size of a data unit only depends on the number of outstanding data accesses to be served and the average performance of a disk (positioning time and transfer rate).

There are a number of variations of simple striping which try to break the very regular structure of this approach. One generalization is *staggered striping* [Ju94] in which a logical clustering of disk drives is implemented. Instead of arranging the data units into clusters of disk drives according to a needed bandwidth they are logically distributed over the whole network. Staggered striping defines a stride s by which consecutive data units are separated, e.g. $s = 1$ defines simple striping. This gains disk utilization if data elements of different bandwidth requirements are accessed.

A different approach is followed by *randomized distribution schemes* [Tha97, Bir01, RN00]. In general, a data unit b is placed on a randomly chosen disk D_i at a randomly chosen position. In practice, randomization is implemented by pseudo-random hash functions or pre-computed random distributions [Mun98a]. This makes not only randomization feasible but gives an efficient access scheme, too. A bound for the load of each disk can then be determined by the balls-into-bins model (e.g. see [Ste98] for an overview). For the case that the number of data units is greater than $n \log n$ the load of each bin has the order of the mean value (with high probability). In particular, there exists a bin that receives $m/n + \Theta(\sqrt{m \ln n / n})$ balls with high probability (w.h.p.). Surprisingly, the multiple-choice case (each ball has the random choice of $d \geq 2$ bins) behaves differently [Vöc00], i.e. the load does not decline with increasing m. The authors have shown by using coupling Markov Chains that the load is independent from the number of balls m. Their simple algorithm works as follows. A number of m balls are sequentially processed. Each ball has the choice of $d \geq 2$ different bins and picks the one with the smallest load.

A slightly different approach is followed by Sanders [Kor00]. His goal is to access any t disk units as fast as possible, i.e. with only t/n parallel access operations. Given a buffer space of size $\mathcal{O}(n/\epsilon)$ (up to $(1-\epsilon) \cdot n$ requests have to be handled in every access step) which hosts n writing queues and given that every data unit is replicated and put on a randomly chosen disk, any t data units can be accessed with only $\lceil t/n \rceil + 1$ access operations. This is done by deriving an optimal parallel schedule using maximum flow computations.

It can be said that the simple striping techniques are sufficient to achieve an optimal space and access balance as long as the system is static (the number of disks does not change over time) and the access pattern is balanced (no hot spots). Nevertheless, the randomized approaches are closing the gab on deterministic approaches in static settings fast and they possess many advantages when it comes to dynamic systems and irregular access patterns. Unfortunately, most of the randomized solutions work with ideal randomization which is impossible to implement. So, there is a large demand for practical implementations of randomized approaches. Due to their simple structure, deterministic solutions are currently favored in practice.

4 Availability

Availability describes the property to retrieve the data units even in the case of failed disk drives and, hence, lost data units. It is a crucial feature of storage networks because the probability of a failure in a collection of disks is scaled by its size [Kat88].

The problem of availability can be solved by using *redundancy*. The easiest way is to store c copies of all data units like in an RAID 1 system (with $c = 1$, which is called mirroring in the storage community) [Kat88] or the PAST storage server [Dru01, Row01]. Naturally, the distribution scheme has to ensure that the copies are hosted by different disks. The system can tolerate up to $c - 1$ failures and is still operational (only with degraded performance) so that the faulty disks can be replaced. The data that has been hosted by the faulty disk can then be rebuild using one of the redundant copies.

Because a full replication scheme results not only in the necessity to update all copies in order to keep the scheme consistent but also in a large waste of resources, more space efficient methods are sought. One of the common approaches is the use of *parity information* like in RAID level 4/5 [Kat88, Pat94b], Random RAID [Bir01], Swarm [Har00, Spa99], and many video servers [Ju94, Mun98a, Gan93, Goy95, Tha97]. The idea behind is the use of redundant information to secure more then just one data unit. This can be done by deriving the bit-wise parity information (parity means taking an bit-wise XOR operation) of a whole stripe and storing it in an extra parity unit. Assuming the units reside on different disks, one disk failure can be tolerated. When a disk failure occurs the faulty disk has to be replaced and the unavailable data units must be rebuild by accessing all other units in its stripe. During this reconstruction phase, the next failure would be hazardous. Obviously, this reconstruction should be done as fast as possible aiming to minimize the maximum amount of data one has to read during a reconstruction [Gib92]. This can be done by choosing a stripe length less then n. The resulting reconstruction load for each disk is given by the declustering ratio $\alpha = \frac{l-1}{n-1}$. If $\alpha = 1$ (like in the RAID 5 layout) each of every surviving disks participates in the reconstruction.

Using redundancy imposes a new challenge to the data distribution because it has not only to ensure an even distribution of data units but also an even distribution of parity information. The reason for this lies in the different access pattern for data units and parity units. First of all, we have to distinguish between read and write accesses. A read can be done by getting any l units. A write operation to any data unit (call it *small write*) has not only to access the written data unit but also the parity unit because the parity information after a write may have changed. It follows that the access pattern to the parity unit is different from all other data accesses. There are a number of different approaches to handle this situation. In RAID level 4 [Kat88] the parity units are all mapped onto the same disk. As long as the whole stripe is written (call it *full write*) this does not impose any problems. Unfortunately, this is not always the case. Small writes will put a lot of stress on the disk storing the parity information. A solution to this is provided by RAID level 5. Here, the parity units are spread over all participating disks by permutating the position of the parity unit inside each stripe. In the ith stripe the parity unit is at position $i \bmod l$.

As long as the number of disks is equal to the stripe length l this approach distributes the parity units evenly. The more general case, allowing the length to differ from n, calls for another distribution scheme like *parity declustering* [Gib92]. The authors propose the use of complete and incomplete block designs. A *block design* is the arrangement of ν distinct objects into b blocks or tuples, each containing k elements, such that each object appears in exactly t blocks, and each pair of objects appears in exactly λ_p blocks. For a complete block design one includes all combinations of exactly k distinct elements from a set of ν objects. Note, that there are $\binom{\nu}{k}$ of these different combinations. Furthermore, only three variables are free because $b \cdot k = \nu \cdot r$ and $r(k-1) = \lambda_p(\nu - 1)$ are always true.

If we now associate the blocks with the disk stripes and the objects with disks the resulting layout (called block design table) distributes the stripes evenly over the n disks (i.e. ν equals the length of a stripe l and k equals the number of disks

n). But such a block design table gives no information about the placement of the redundant information. To balance the parity over the whole array we build l different block design tables putting the parity unit in each of them at a different position in the stripe. The full (or complete) block design is derived by fusing these block design tables together. Obviously, this approach is unacceptable if the number of disks becomes large relative to the stripe length l, e.g. a full 41 disk array with stripe length 5 has 3,750,000 blocks. Hence, we have to look for small block designs of n objects with a block size of l, called balanced incomplete block designs (BIBDs), which do not need full tables to balance the parity units. There is no known way to derive such designs algorithmically and for all possible combinations of parameters a BIBD might not be known. Hall [Hal67] presents a large number of them and some theoretical techniques to find them. In this case, the use of complete designs or the use of the next possible combination is recommended.

The approach of using parity information can be extended to tolerate more then one disk failure. As an example the EVENODD layout [Men94] can survive two disk failures. This is done by taking the regular RAID 5 layout and adding parity information for each diagonal on a separate disk. Such an approach can be even extended to tolerate t arbitrary simultaneous disk failures. Gibson et al. [Pat94a] showed that in this case, $t \cdot l^{1-\frac{1}{t}}$ disks worth of parity information are needed. Taking fast reconstruction into account, these schemes are only of theoretical interest.

The above mentioned techniques try to solve different problems, like availability and space balance. Combining them leads to new and more general data distributions. In [Sut99, Sut96] the EVENODD layout is used together with the parity declustering technique. First, an appropriate BIBD is chosen. The data units in each stripe in this layout are then replaced by a complete EVENODD layout. To do so, the data units of the BIBD are partitioned such that they will host a full column of an EVENODD layout. Such a data distribution can tolerate two disk failures and allows faster reconstruction compared to RAID 5 layouts. Furthermore, the authors reduce the problem of finding a data distribution scheme to the partitioning of the storage network into parity stripes (defining the length l and the replication factor r). This is done by using a network flow based method. The idea is further extended in [Hol96] by defining approximately balanced distribution schemes. The major gain lies in the construction of such layouts. The authors propose three different construction methods: randomization, simulated annealing, and perturbation of exact layouts. They implemented a simulation environment and tested the behavior in degraded mode (when one disk is faulty) and reconstruction mode. As a result, the performance of approximately balanced layouts is very close to the performance of BIBDs based layouts.

The redundant information used to protect the storage network against disk losses can also be exploited to decrease the data access time significantly. In the case of $c \geq 1$ copies a simple protocol might work as follows. Suppose, there exists information about the queue length of the disk drives., e.g. gathered by probing. Because each data element has at least two copies, we always access the disk with the smallest queue length. Using the same arguments as in [Ste98] it can be shown, that accesses are very evenly balanced.

In [Bir01] the same technique is applied to parity extended random distribution schemes. As noted above, any $c-1$ units suffice to reconstruct the complete stripe. Hence, a data access might choose those data units which reside on the least loaded disks. By similar arguments as in [Ste98] this should improve the load substantially. Note, that some further processing is needed if the parity information is accessed. This approach is especially interesting for random distributions because it clips the tail of the disk access queue-length distribution. The author has implemented some simulations and gives evidence that such a strategy improves the response time significantly.

5 Heterogeneity

Heterogeneity describes the ability to handle disks with different characteristics efficiently, i.e. use the disk according to its capacity or its bandwidth. This problem is easy to solve if all other properties are neglected but is rather challenging if the space and access balance must also be guaranteed.

Heterogeneity is becoming more and more common in recent years. First of all, with the growth of the data volume, storage systems quickly run out of space. If a complete exchange of the system is unacceptable (due to the higher cost) one has to expand the existing configuration. It is rather unlikely that disks with the same characteristics can be found which results in a heterogeneous storage network. Furthermore, the larger the range of the underlying network (ranging form SCSI buses to LAN and WAN) the more likely heterogeneity becomes. As an example, look at clusters of workstations in a company. Suppose, there is a dedicated workstation pool that is connected via a fast network. As a result to modernization, the fast network may now span the whole company expanding the capabilities of the cluster to every single computer. To use the full potential of such an investment, the cluster must be capable of handling different workstations with different characteristics. Furthermore, it is rather unlikely that a cluster of workstations is completely exchanged by a new one. This too, leads to a heterogeneous system.

A further field of interest are *peer-to-peer* networks (P2P). Such a network consists of a changing number of servers (peers) which work together to manage special tasks like storing of, distributing of, and searching for data elements. These tasks are solved without a centralized server. Hence, each peer only knows its next neighbors (or some part of the network). When some new server wants to join the network, it contacts some server inside and joins its neighborhood. P2P networks usually use some established network to communicate. In other words, their are overlay networks that provide their services on top of basic communication. Because there is no central instance to supervise the joining or leaving of nodes, the general structure of a P2P network is very likely to be heterogeneous concerning capacity and bandwidth.

Until recently, little attention was paid to the heterogeneity problem, e.g. implementations of the RAID distribution schemes can work over different sized disks but they simply treat each disk as if it has the size of the smallest disk which results in a waste of space capacity.

Given a number of different disk drives, an easy and quick solution to the problem is the clustering of disks according to their characteristics [Sha93]. Especially in multimedia systems, this is a common technique due to the bandwidth requirements in such systems. Within a cluster, the placement of data units can be done optimally with aforementioned techniques (namely striping). The big advantage of this approach is its simple extendibility. When new disks with new characteristics enter the system they simply form a new cluster. But the clustering approach has a main disadvantage. Adding new (and usually faster) disks does not improve the response time of all data units. Only the access to data elements hosted by newer disks will be significantly faster. This implies also that response time depends on the cluster a data element is hosted by. Assuming that the access pattern is known in advance, this could be used to avoid hot spots by putting popular data elements on faster disks. But unfortunately, the access pattern is usually unknown and may change significantly over time.

Another approach to handle heterogeneous disks was proposed by Sanders [San01]. Here, the aim is to design optimal scheduling algorithms for arbitrary data requests. Each data element is replicated and randomly mapped to separated disks. Then, the scheduling problem can be transformed into a network flow problem. The requests are on the left hand side (source) and have an edge (with weight ∞) to all disks which store a copy of the requested data element. On the right hand side of the flow network (sink), the disks might be served by disk controllers and I/O-busses each of which is modeled by an edge with an appropriate weight to the disks/controllers it connects. Now, the flow problem can be solved resulting in a scheduling algorithm for the data requests. Heterogeneity can be introduced by adjusting the edge weights of the flow network according to the given capacities. The disadvantage of this approach lies in its batch-like behavior. For any collection of new requests a new flow problem must be solved. Especially for large systems the number of needed requests is considerable.

In the next four sections we will introduce new approaches that try to solve the heterogeneity problem.

5.1 AdaptRaid

AdaptRaid [Lab00, Lab01] is a distribution scheme that extends the general RAID layout so that heterogeneity (corresponding to different capacities) can be handled. The basic idea is very simple. Larger disks are usually newer once and, thus, should serve more requests. Nevertheless, the placement of whole stripes should be kept as long as possible to gain from parallelism. So far, there are extensions to handle the case without data replication (RAID level 0) and with fully distributed parity information (RAID level 5).

AdaptRaid level 0 The initial idea of AdaptRaid level 0 [Lab00] is to put as many stripes of length $l = n$ on all disks as possible. As soon as some k disks cannot store any more data units, stripes of length $n - k$ are mapped onto the remaining $n - k$ disks. This process is repeated until all disks are full (see Fig. 2).

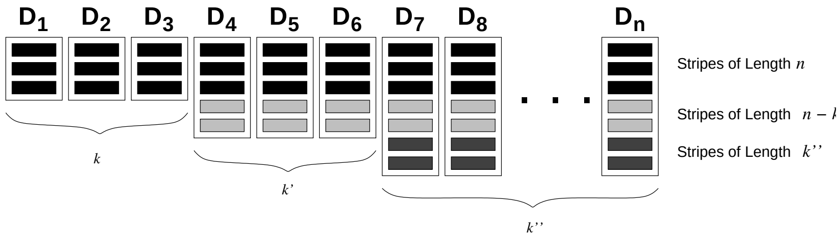

Fig. 2. The initial idea is to place as many stripes over all disks as there is capacity in the k smallest ones. When they saturate this strategy is repeat with the $n - k$ disks.

This approach has the major drawback that the access to data stored in the upper part of the disk is faster then the access to data in the stored in the lower part of the disk drives because of the different stripe length. Furthermore, the assumption that newer disks are faster is in general not true. The following generalization will cope with both problems. First, we define the utilization factor $UF \in [0, 1]$ to be the ratio between the capacity of a disk and its bandwidth. This factor is determined on a per disk basis, where the largest disk has always $UF_i = 1$ and all other disks get a UF according to their capacity and relative to the largest one. These values should be set by the system administrator. To overcome the access problem, data patterns, which capture the overall capacity configuration, are defined. These patterns have a similar structure to the pattern of the initial idea, but are kept smaller such that many of them can fit on a single disk. The size of these patterns is measured by the second parameter – lines in a pattern (LIP). In general, this parameter is an indicator for the distribution quality of the layout and is measured by the number of units hosted on the largest disk. The overall data layout is defined by repeating these data patterns until the disks are full (see Fig. 3).

The access to data elements is twofold. In the first phase, the correct pattern has to be found which can easily be done because the size of a pattern (in number of units) is known. Then, the requested data unit is found by accessing the pattern itself. The space resources needed to access any data element is proportional to the size of the pattern.

The performance of AdapRaid0 was tested using a disk array simulator. The tests have been made with varying ratios of fast and slow disks[2]. Two different scenarios have been tested. First, the performance is compared to a RAID level

[2] The faster disks possess roughly doubled performance parameters, like average seek time, short seek time, and local cache.

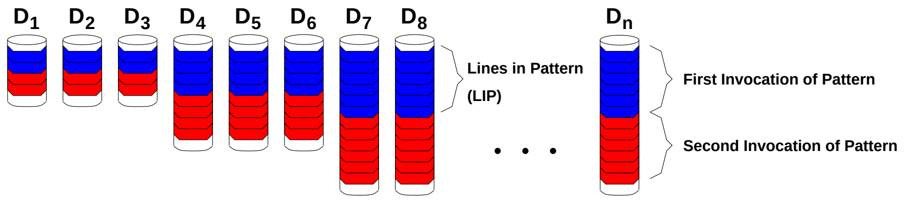

Fig. 3. Using the pattern defined by the initial idea, one distributes longer lines over the whole disk by fusing many invocations of the pattern on the disk.

0. Naturally, AdaptRaid0 can shift the load from slower disks to the faster ones resulting in a performance gain between 8% and 20% for read accesses and 15% up to 35% for write accesses. The second scenario compares AdaptRaid0 to a configuration that contains only fast disks. This is reasonable because it gives an evaluation of the usefulness of keeping older disks in the system and therefore of the effect of parallel disk accesses. The experiments show that the performance gain is more pronounced in read accesses, e.g. in an array with 2 fast and 6 slow disks the read accesses can be answered 10% faster then in a fast disk only configuration.

It can be concluded that the exploitation of heterogeneity gains a significant performance speed-up.

AdaptRaid level 5 AdaptRaid level 5 [Lab01] is the heterogeneous variant of RAID level 5. It uses the same simple idea as AdaptRaid0, mainly to put more data units on larger and usually faster disks. But here, the distribution problem is more complex because to guarantee a fast reconstruction time the parity information must to be evenly distributed, too.

Following the same approach as in AdaptRaid0 and keeping the parity distribution of RAID 5 will result in a small write problem. As can be seen in Fig. 4, such a layout has stripes of different length. For the distribution scheme this is not a problem, but when it comes to performance the situation is different. Note, that the parity unit needs to be updated every time a data unit is written. Hence, file systems (see Chapter **??**) should group the data requests such that full stripes could be written as often as possible. Such a technique will avoid many small writes and works well when the stripe size is fixed. Therefore, such optimizations would be useless for the proposed data layout.

This problem can be avoided by restricting the stripe length of each stripe to integer divisors of the length of the largest stripe (see the leftmost layout in Fig. 5). This leads to a heterogeneous layout which may waste some capacity. But a careful distribution of the free units can significantly improved this scheme. First, the layout should be independent of the number of disks still participating. This can be achieved by letting each stripe start on a different disk (see the center layout of Fig. 5). The nice effect of this transformation is the even distribution of the wasted free space over all disks instead of concentrating them on a few

Fig. 4. The left hand side shows the regular RAID 5 layout. Parity units are distributed over the whole array by letting every stripe start on a different disk in a round-robin manner. The right hand side shows the initial layout in AdaptRaid5. Here, the RAID 5 strategy is applied on stripes of varying size.

disk drives. Now, we can get rid of the wasted capacity by using a 'Tetris' like algorithm. The holes of free units are just filled with subsequent data units from stripes further below (data units form these stripes 'move' upwards). Because the free spaces are distributed over all disks whole stripes can be added at the end of the pattern. Still the layout is regular and only the size of one pattern is need to locate any data unit (see the rightmost picture in Fig. 5). As in the AdaptRaid0, we will generalize the solution by defining two parameters. As before, UF is the utilization factor describing the load of a disk related to the largest disk. Similar to LIP the parameter SIP – stripes in pattern – controls the distribution of pattern over the whole array.

For the practical evaluation, the same simulator as for AdaptRaid0 has been used. The new approach was tested against a RAID 5 and a homogeneous configuration of only fast disk[3]. Obviously, both approaches waste some capacity in one way or the other. The performance was measured for reads, small writes and full writes (writing a full stripe). It was observed [Lab01], that AdaptRaid5 is almost always the best choice and scales gradually with an increasing number of fast disks. Only when large chunks of data are read, the method is outperformed by the homogeneous configuration of only fast disks. This is due to the fact that the slower disks cannot deliver the data units appropriately. When full stripes are read they are the bottleneck of the system. Nevertheless, when simulations of real workloads are used, the performance gain of the new approach can be around 30% if only half of the disks are fast disks.

5.2 HERA – Heterogeneous Extension to RAID

A different kind of heterogeneity is used in the disk merging approach [Gha00]. Here, the aim is not only to provide heterogeneous bandwidth distributions but also to increase the reliability of the system.

The disk merging approach constructs a homogeneous, logical view out of a physical collection of disks. Each physical disk D_i is partitioned into $k_i \in \mathbb{R}$ equal sized *extents*. These extents are arranged into G parity groups – the logical view

[3] In the experiments, the number of fast disks varies from 2 to 9 disks.

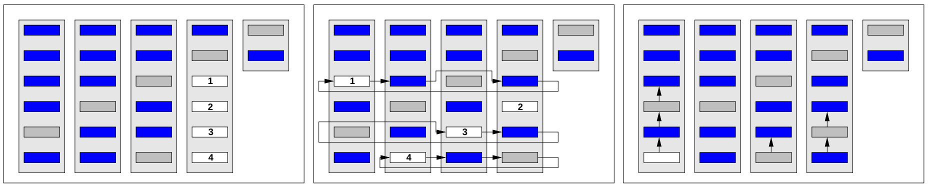

Fig. 5. The left figure shows a placement, so that the length of any stripe is a divisor of the length of the largest stripe. Clearly, some disk may have unused capacity. In the next step, the free space is distributed over all disks of this stripe length as shown in the center. This opens the possibility to apply an 'Tetris'-like algorithm. The free spaces are erased by moving the subsequent data units on each disk upwards. The resulting layout is depict in the right figure. The arrows indicate the movement of data units.

of the system. If the parity groups are fused together they form a homogeneous system where the stripe length l is the number of extents in a parity group. It is possible that an extent gets capacity and bandwidth from different physical disks. The data distribution scheme simply stripes the data units over the parity groups (see Fig. 6).

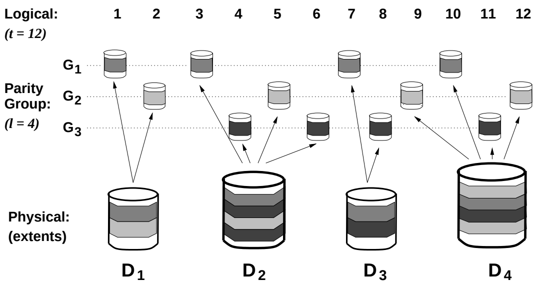

Fig. 6. HERA divides the capacity of a disk into logical extents $1, \ldots, D^t$. Each physical disk D_i has capacity for k_i many logical disks. The logical disks are arranged in G parity groups each of which appears homogeneous.

How many parity groups are possible? In order to ensure the fault tolerance of the system enforced by the used striping strategy, each extent in every parity group has to come from a different physical disk. Let D^t denote the number of logical extents in the system. Then, the number of parity groups has to fulfill the following inequality:

$$G \leq \left\lfloor i \frac{D^t}{k_i} \right\rfloor \qquad 0 \leq i \leq n$$

To show the reliability of the system, the behavior is modeled by different processes like the failure process and the reconstruction process. Because the mean time to failure of any disk can be estimated, the time of these processes can be estimated, too. Using a Markov model [Gib91], the mean time to service loss (MTTSL) can be derived for such a heterogeneous system. It was shown that the reliability (measured in MTTSL) is only a factor of approximately 10 away from the best possible configuration, namely the clustering of identical disks.

Nevertheless, this approach still needs the experienced hand of an administrator and the performance strongly relies on his decisions.

5.3 RIO – Random I/O Mediaserver

The RIO mediaserver [Mun98b, Mun98a] was build as a generic multimedia storage system capable of efficient, concurrent retrieval of many types of media objects. It defines a randomized distribution strategy and supports real-time data delivery with statistic delay guarantees. The used randomized distribution scheme is very simple. The data units are placed on a randomly chosen disk at a randomly chosen position[4] (see Fig. 7). Heterogeneity may occur in different disk capacities and varying bandwidth properties. As noted before randomization ensures a good balance in the long run, especially when it is used in conjunction with replication. In addition, RIO also exploits redundancy to improve the short time balance by carefully scheduling the accesses to data units.

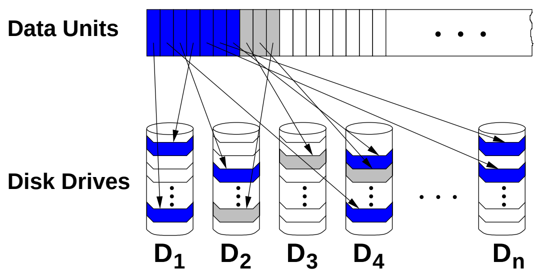

Fig. 7. RIO assigns every data unit b to a random position on a randomly chosen disk.

But randomization can also be used to handle heterogeneity. The initial idea is to put more data units on larger disks. This can be done as follows. Let $d_j = \frac{C_j}{S}$ be the relative (normalized) capacity of disk D_j. Note, that $\sum_{1 \le j \le n} d_j = 1$. Now d_j is used as the probability of putting a data unit on disk D_j. In the long run this would fill the disks according to their capacity. Nevertheless, there might be short time imbalances and the bandwidth problem is not taken into account.

How can both, capacity and bandwidth, be optimized simultaneously? The simplest way is a coupling of bandwidth and capacity by defining the bandwidth

[4] The latter is only important for the theoretical analysis. A practical implementation can only ensure pseudo-random behavior.

to space ratio $BSR = \frac{b_i}{C_i}$. The higher the BSP the faster the disk can deliver all its data units, hence, the more data units should be mapped to these disks. Obviously, the disks with low BSR present the bottleneck in the system. The major idea for a better balance is to shift some load from disks with low BSR towards disks with high BSR. This can be done using replication, and putting the copies on disks with high BSR. But how much redundancy is needed to sustain a given load? This question can be answered by using the following approach. First, the disks are grouped into t clusters according to their BSR and sorted in ascending order. Let N_i be the number of disks in cluster i. Further let $S_i = \sum_{1 \leq j \leq N_i} C_i$ and $B_i = \sum_{1 \leq j \leq N_i} b_j$ be the accumulated capacity and bandwidth in cluster i, respectively. Let S and B define the total capacity and bandwidth in the storage network and let $S'_i = S_i/S$ and $B'_i = B_i/B$ be the relative capacity and relative bandwidth of cluster i, respectively (see Fig. 8). The relative bandwidth space ratio can then be defined as $BS_i = B'_i/S'_i$. For a homogeneous system, $BS_i = 1$. The cluster with the lowest BS_i will be the most stressed cluster, not only because its disks have the slowest bandwidth but also because they host a large chunk of the data units. The number of data units that can be stored in the system is defined by $U = \frac{S}{1+r}$ where r denotes the replication rate. In formal terms, we want to use replication to shift the stress from clusters with $BS_i < 1$ to clusters with $BS_i > 1$. But how much replication is needed to sustain a certain maximal load $\lambda_{max} \leq B$?

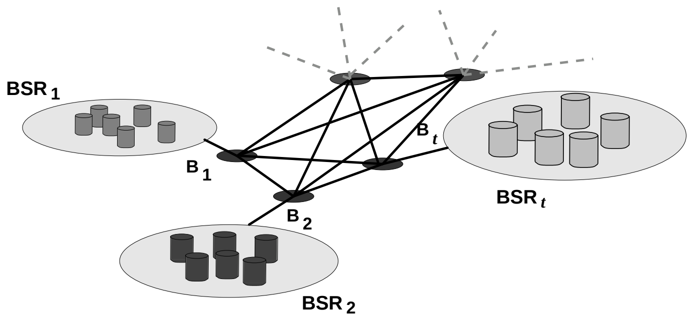

Fig. 8. RIO groups the disks according to their BSR. Each of the t groups is homogeneous in itself and has a collected capacity of S_i and a collective bandwidth of B_i.

To answer this question, we have to distinguish between two different cases. Let $\rho = \lambda_{max}/B$ be the average disk utilization. If $\rho = 1$ we have full utilization and if $\rho < 1$ the disks are only partially utilized. Lets consider full utilization first. It can only be achieved, if the load directed to a cluster i is in order of its bandwidth B_i. The maximum load that can be directed to any cluster i is reached when all data units stored in this cluster are distinct and all accesses to these data units are send to i. Hence, the maximal load can be computed by

$\lambda_i = \lambda_{max} \cdot \frac{S_i}{U} = \rho \cdot B \cdot (1+r) \cdot S_i'$. To use all the available disk bandwidth on the system, we have to fulfill $\lambda_{i_{max}} \geq B_i$. This is satisfied if $BS_i \leq (1+r)$ for all $1 \leq i \leq m$. It follows that the replication rate can be expressed by $r \geq \max_i\{BS_i\} - 1 = BS_m - 1$.

In a similar way, one can determine the minimum replication rate to ensure partial bandwidth utilization. Furthermore, it can be shown that this replication rate is sufficient to achieve the desired utilization by defining a distribution scheme mapping replicated data units to clusters according to their BSR.

6 Adaptivity

One motivation for the heterogeneity in Sect. 5 was the addition of newer disks due to a growing space demand. But how does the introduction of new disks effect the still operational system? Obviously, if we want to have better performance when new disks are added, the data units must be evenly distributed over all disks. Hence, some portion of the existing data units has to be redistributed. The efficiency of this process is described by *adaptivity*. A distribution scheme is adaptive if it only has to redistribute a small number of data units in case of a changing system. Such a change can be the entering or failing of disks or the introduction of newly accessed data units. To capture heterogeneous requirements, we define the property of *faithfulness*. A distribution scheme is faithful if after a change in the system it can always ensure that the number of data units stored on a disk is according to its capacity/bandwidth requirement. In the homogeneous case each disk should get the same number of data units. For the more general case, lets denote by d_i the relative capacity of each disk D_i (relative to the total capacity S). Then a distribution scheme is faithful if it can ensure that any disk D_i gets $(d_i + \epsilon) \cdot m$ data units for an arbitrary small ϵ. Faithfulness uses the ϵ-term to allow randomized approaches to possess this property. Note, that m denotes the number of data units currently in the system.

The adaptivity can be measured by *competitive analysis*. For any sequence of operations σ that represent changes in the system, we intend to compare the number of (re-)placements of data units performed by the given scheme with the number of (re-)placements of data units performed by an optimal strategy that ensures faithfulness. A placement strategy will be called *c-competitive* if for any sequence of changes σ, it requires the (re-)placement of (an expected number of) at most c times the number of data units an optimal adaptive and perfectly faithful strategy would need.

Looking at the distribution techniques of the last chapters, e.g. the striping algorithm and the random distribution, none of them is adaptive. Consider a stripe of length n. Introducing a new disk literally changes the whole distribution. If only the $(n+1)$ fraction of data units are redistributed, the system needs to keep track of the redistributions and, in the long run, to store every position of any data unit to access it. A similar argument can be used for random distributions.

6.1 Consistent Hashing

Consistent hashing [Pan97, Lew98] was developed for distributed web caching but can also be applied here. The algorithm overcomes the drawback of conventional hashing schemes by defining a hash function that is consistent in case of a changing number of servers. We use the balls-into-bin analogy for the description. The algorithm works as follows.

Suppose we are given a random function f_B and a set of independent, random functions $g_1, \ldots, g_k$, where k may depend on n, the number of bins. The function $f_B : \{1, \ldots, M\} \to [0, 1)$ maps the balls uniformly at random to real numbers in the interval $[0, 1)$ and each function $g_i : \{1, \ldots, N\} \to [0, 1)$ maps the bins uniformly at random to real numbers in the interval $[0, 1)$. A ball i is assigned to the bin closest to it, regarding to this mapping and viewing $[0, 1)$ as a ring, i.e. it is assigned to bin D_i that minimizes $\min_j \min[|f_B(i)-g_j(D_i)|, 1-|f_B(i)-g_j(D_i)|]$.

From the proofs in [Pan97] it follows that this strategy is faithful and 2-competitive (in the expected sense, if renaming of bins is not allowed) and that it can be implemented in a way that the location of a ball can be determined in expected constant time. It requires k to be in order of $\Omega(\log N)$ to ensure that the number of balls differs only by a constant factor from the expected number with high probability. This results in a space consumption of around $\mathcal{O}(n \log N)$ words (assuming that one word can hold $\log n$ bits and not including the hash functions).

This strategy does work well in an homogeneous system. One might think that it can be easily extended to cover the heterogeneous case by allowing disks with more capacity to have more random points in $[0, 1)$. However, this would require $\Omega(\min[C_{\max}/C_{\min}, m])$ points to be faithful, where $C_{\max}$ is the maximum capacity and $C_{\min}$ is the minimum capacity of all disk. Thus, in the worst case, the number of points could be as much as $\Theta(m)$, violating severely the space consumption property. On the other hand, restricting the total number of points to something strictly below m cannot guarantee faithfulness under any capacity distribution (just consider two disks with capacities c/m and $(m-c)/m$ for some constant $c > 1$).

6.2 Cut-and-Paste Strategy

A two-phase approach is used by the cut-and-paste strategy [Sch00]. We will stick to the balls-into-bins model (data units are represented by balls and disk drives are represented by bin, respectively) to describe the strategy.

Due to the fact that the number of data units is not fixed in advance[5] we define an address space of $\{1, \ldots, p\}$ for the balls. The parameter p is arbitrarily large but fixed (for current systems 2^{64} will suffice). The first phases maps all possible balls M to a $[0, 1)$ interval by using a random hash function $h : M \to [0, 1)$ (see the first picture in Fig. 9). This has the effect that any number of m

[5] The adaptivity allows the a storage system to grow dynamically. Hence, the possible number of accessed data units is not restricted by the overall capacity of the network.

balls are evenly distributed over the $[0, 1)$ interval. Hence, it is sufficient to assign equal sized shares of the interval to the bins to ensure a certain even distribution. Initially, the hash value $h(b)$ denotes the height (in the $[0, 1)$ interval) of that ball. The second phase now makes certain that all bins get an even share. This is done by cutting the $[0, 1)$ interval into ranges and these ranges (and therefore all the balls that fall into them) are mapped to bins by an *assimilation function*. If more then one range is assigned to a bin, they are fused together and adjusted such that the range on any bin is continuous and starts with 0. The height of the balls falling into the ranges are adjusted accordingly. Let $[0, 1/n]_i$ denote the range mapped to bin i when n bins are currently in the system. The assimilation function maintains the invariant that the (accumulated) ranges have the same size. Furthermore, it ensures that only a small portion of every range is remapped if the number of bins changes. The definition of the assimilation function is repetitive in the number of bins. It starts by assigning the interval $[0, 1)$ to bin 1. Then, the change from n to $n + 1$ bins is defined by cutting off the range $[1/(n + 1), 1/n]_i$ from every bin i with $i \in \{1, \ldots, n\}$ and fuse them together such that they are in reversed order (the range coming from bin 1 is topmost). This will be the new range for bin $n + 1$ (see also second picture in Fig. 9). Due to the fact that each bin gets an equal share of the $[0, 1)$ interval it gets also an almost equal number of balls, accuracy depending only on the hash function. Furthermore, the number of balls moved to a new bin comes evenly from all other bins and is in the order of the balls needed to be moved to ensure an even distribution.

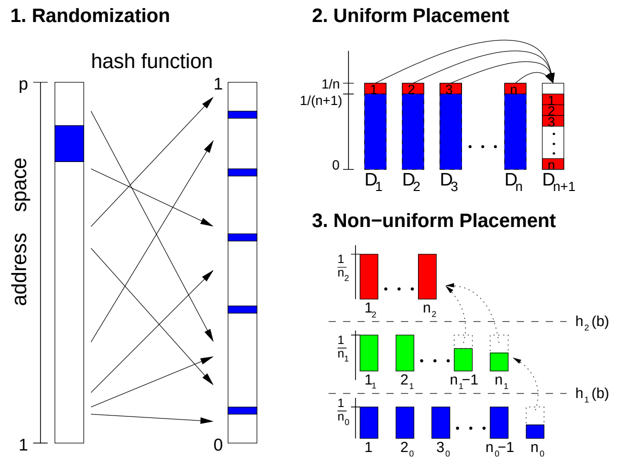

Fig. 9. The cut-and-paste strategy. First, the address space is hashed into a $[0, 1)$ interval. Then the uniform placement is done according to an assimilation function which defines the transition from n bins to $n + 1$ bins. Heterogeneity is achieved by defining a number of levels in which only disks with free capacity participate.

This strategy will not only ensure the fast adaptation to a changed number of bins but also provide an efficient way to access the balls. It suffices to trace the replacements performed by a ball for the consecutive addition of 1 to n bins. This can be done by determining the condition whenever a ball is replaced. In other words, when the height of the ball in the range will be greater then $1/(n+1)$. It can be shown that only $\log n + 2$ replacements will occur and therefore only a logarithmic number of operation is necessary to access any ball.

Unfortunately, this approach can not be easily extended to the heterogeneous case. The authors propose the introduction of levels in which only the bins with free capacity participate. But unfortunately, the competitive ratio depends logarithmically on the number of balls currently in the system which is rather inefficient.

Recently, both of the above strategies were extended to meet either the heterogeneity and the adaptivity requirements [Sch02]. They propose two new strategies, called SHARE and SIEVE, that are perfectly faithful and 2-competitive. Both strategies are based on random hashing and need only a logarithmic number of steps to access any data element (w.h.p.).

7 Conclusions

The constantly growing amount of data calls for flexible and efficient storage systems. It has been shown that storage networks can keep up to this expectation. There is a large number of different techniques to exploit the inherent parallelism of such networks to gain better performance. Especially smaller systems like RAID arrays or multimedia data servers can be used very efficiently.

Nevertheless, the systems are growing rather rapidly. This introduces new problems like heterogeneity, adaptivity, or the resource efficiency considerations. For these problems there exist only a few approaches presented in this paper. Most of them have not been practically tested, yet.

In the last few year two very interesting application fields emerged. The first is the area of Storage Area Networks (SAN). The fast growing data volume especially in enterprises leads to very high cost of data maintenance. This cost can be significantly reduced if a centralized storage system is used. Nevertheless, such a storage system must allow for a growing data volume and they are very likely to be heterogeneous. Carefully managed storage networks are capable of providing these features.

The second new field of interest are peer-to-peer networks (P2P). First of all, these networks are highly heterogeneous (with respect to space and connection requirements). Furthermore, availability is a big issue in these overlay networks. Data will be stored in them and the main task will be to retrieve the data. Furthermore, a P2P network is highly dynamic. Server will join and leave it without further notice calling to efficient adaptivity.

For all these reasons, techniques used in storage networks will become more popular in the near future.

References

[Bir01] Y. Birk. Random RAIDs with selective exploitation of redundancy for high performance video servers. *Readings in multimedia computing and networking*, pages 671–681, 2001.

[Dru01] A. Rowstron, P. Druschel. Storage management and caching in PAST, A large-scale, persistent peer-to-peer storage utility. In Greg Ganger, editor, *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP-01)*, volume 35, 5 of *ACM SIGOPS Operating Systems Review*, pages 188–201. ACM Press, 2001.

[Gan93] F.A. Tobagi, J. Pang, R. Baird, M. Gang. Streaming RAID: A disk array management system for video files. In *Proceedings of the First ACM International Conference on Multimedia*, pages 393–400. ACM Press, 1993.

[Gha00] R. Zimmermann, S. Ghandeharizadeh. HERA: Heterogeneous extension of raid. In H.R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 4, pages 2159–2165. CSREA Press, 2000.

[Gib91] G.A. Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage.* PhD thesis, University of California at Berkeley, December 1991.

[Gib92] M. Holland, G.A. Gibson. Parity declustering for continuous operation in redundant disk arrays. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, volume 27(9), pages 23–35. ACM Press, 1992.

[GM86] K. Salem, H. Garcia-Molina. Disk Striping. In *Proceedings of the 2nd International Conference on Data Engineering*, pages 336–342. ACM Press, 1986.

[Goy95] H.M. Vin, S.S. Rao, P. Goyal. Optimizing the placement of multimedia objects on disk arrays. In *International Conference on Multimedia Computing and Systems*, pages 158–166. IEEE Computer Society, 1995.

[Hal67] M. Hall Jr. *Combinatorial Theory.* Blaisdell Publishing Company, 1967.

[Har00] I. Murdock, J.H. Hartman. Swarm: A log-structured storage system for linux. In *Proceedings of the FREENIX Track: 2000 USENIX Annual Technical Conference*, pages 1–10. USENIX Association, 2000.

[Hol96] E.J. Schwabe, I.M. Sutherland, B.K. Holmer. Evaluating approximately balanced parity-declustered data layouts for disk arrays. In *Proceedings of the Fourth Workshop on Input/Output in Parallel and Distributed Systems*, pages 41–54. ACM Press, 1996.

[Joh84] O.G. Johnson. Three-dimensional wave equation computation on vector computers. In *Special issue – Supercomputers - Their Impact on Science and Technology computational physics*, volume 72(1), pages 90–95. IEEE Computer Society Press, 1984.

[Ju94] S. Berson, R. Muntz, S. Ghandeharizadeh, X. Ju. Staggered striping in multimedia information systems. *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, 23(2):79–90, 1994.

[Kat88] D.A. Patterson, G.A. Gibson, R.H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of Association for Computing Machinery Special Interest Group on Management of Data: 1988 Annual Conference*, pages 109–116. ACM Press, 1988.

[Kor00] P. Sanders, S. Egner, J.H.M. Korst. Fast concurrent access to parallel disks. In *Proceedings of the Eleventh Annual Symposium on Discrete Algorithms*, pages 849–858. ACM Press, 2000.

[Lab00] T. Cortes, J. Labarta. A case for heterogenenous disk arrays. In *Proceedings of the International Conference on Cluster Computing*, pages 319–325. IEEE Computer Society Press, 2000.

[Lab01] T. Cortes, J. Labarta. Extending heterogeneity to RAID level 5. In *Proceedings of the USENIX 2001*, pages 119–132. USENIX Association, 2001.

[Lew98] D.M. Lewin. Consistent hashing and random trees: Algorithms for caching in distributed networks. Master's thesis, Department of Electrical Engeneering and Computer Science, Massachusetts Institute of Technology, 1998.

[Men94] M. Blaum, J. Brady, J. Bruck, 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, 1994.

[Mun98a] J.R. Santos, R.R. Muntz. Performance analysis of the RIO multimedia storage system with heterogeneous disk configuration. In *Proceedings of the Sixth International Conference on Multimedia*, pages 303–308. ACM Press, 1998.

[Mun98b] J.R. Santos, R.R. Muntz. Using heterogeneous disks on a multimedia storage system with random data allocation. Technical Report 980011, University of California, Los Angeles, Computer Science Department, 1998.

[Pan97] D.R. Karger, E. Lehman, T. Leighton, M. Levine, D.M. Lewin, 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 Symposium on Theory of Computing*, pages 654–663, El Paso, Texas, 1997. ACM Press.

[Pat90] P.M. Chen, D.A. Patterson. Maximizing performance in a striped disk array. In *Proceedings of the 17th Annual Symposium on Computer Architecture, Computer Architecture News*, volume 18(2), pages 322–331. ACM Press, 1990.

[Pat94a] L. Hellerstein, G.A. Gibson, R.M. Karp, R.H. Katz, D.A. Patterson. Coding techniques for handling failures in large disk arrays. *Algorithmica*, 12(2/3):182–208, 1994.

[Pat94b] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, D.A. Patterson. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, 1994.

[RN00] J.R. Santos, R.R. Muntz, B. Ribeiro-Neto. Comparing random data allocation and data striping in multimedia servers. In *Proceedings of the international conference on Measurements and modeling of computer systems*, pages 44–55. ACM Press, 2000.

[Row01] P. Druschel, A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Hot Topics in Operating Systems*, pages 75–80, Schloss Elmau, Germany, 2001. IEEE Computer Society Press.

[San01] P. Sanders. Reconciling simplicity and realism in parallel disk models. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms*, pages 67–76. ACM Press, 2001.

[Sch00] A. Brinkmann, K.A. Salzwedel, C. Scheideler. Efficient, distributed data placement strategies for storage area networks. In *Proceedings of the 12th Annual Symposium on Parallel Algorithms and Architectures*, pages 119–128. ACM Press, 2000.

[Sch02] A. Brinkmann, K.A. Salzwedel, C. Scheideler. Compact, adaptive placement schemes for non-uniform requirements. In *Proceedings of the 14th Annual Symposium on Parallel Algorithms and Architectures*, pages 53–62. ACM Press, 2002.

[Sha93] S. Ghandeharizadeh, C. Shahabi. Management of physical replicas in parallel multimedia information systems. *LNCS, Foundations of Data Organisation and Algorithms 1993*, 730:51–68, 1993.

[Spa99] J.H. Hartman, I. Murdock, T. Spalink. The Swarm scalable storage system. Technical Report TR99-06, The Department of Computer Science, University of Arizona, 1999.

[Ste98] M. Raab, A. Steger. "Balls into Bins" – A simple and tight analysis. In *RANDOM: International Workshop on Randomization and Approximation Techniques in Computer Science*, volume 1518, pages 159–170. LNCS, 1998.

[Sut96] E.J. Schwabe, I.M. Sutherland. Flexible usage of parity storage space in disk arrays. In *8th Annual Symposium on Parallel Algorithms and Architectures*, pages 99–108. ACM Press, 1996.

[Sut99] E.J. Schwabe, I.M. Sutherland. Flexible usage of redundancy in disk arrays. *Theory of Computing Systems*, 32(5):561–587, 1999.

[Swa92] D.P. Siewiorek, R.W. Swarz. *Reliable Computer Systems: Design and Evaluation.* Digital Press, Bedford, Massachusetts, USA, 1992.

[Tha97] J. Alemany, J.S. Thathachar. Random striping news on demand servers. Technical Report TR-97-02-02, University of Washington, Department of Computer Science and Engineering, 1997.

[Vöc00] A. Berenbrink, A. Czumaj, A. Steger, B. Vöcking. Balanced allocations: The heavily loaded case. In *ACM Symposium on Theory of Computing*, pages 745–754. ACM Press, 2000.