Design of the PRESTO Multimedia Storage Network*

Petra Berenbrink† André Brinkmann‡ Christian Scheideler§

October 14, 1999

Abstract

In this paper, we present concepts and simulation results for the design of the **P**aderborn **re**al-time **sto**rage network, short PRESTO, which is currently developed at the Paderborn University in a joint project of the Electrical Engineering Department and the Computer Science Department.

In this project, we aim at constructing a scalable and fault-tolerant storage network that manages a set of parallel disks in a resource efficient way and that can support the real-time delivery of data. We discuss in this paper the principal concepts of the PRESTO storage network, concentrating on data placement and load balancing strategies. Furthermore, we present simulation results that demonstrate that our techniques achieve a high disk utilization together with a low latency with a very high degree of reliability.

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 content found in intranets and the Internet are heavily contributing to the load. As a result, servers and storage are being centralized and consolidated to better manage this explosion of data and the overall cost of ownership. A common approach is to combine storage devices into a dedicated network that is connected to LANs and/or servers. In the following such networks are called *storage networks*. Major requirements for storage networks are scalability, reliability, availability and servicability. Scaleability ensures that today's investment can be used for tomorrow's demands. Reliability protects the system from a loss of data, availability keeps the system operating in spite of a failure of one (or more) of its components, and serviceability helps to set-up and maintain the system. Since the ability to support multimedia data streams is becoming more and more important, several projects have already been set up to study concepts for storage networks that can efficiently handle real-time data requests. See for instance the MARS project [BPC94, BP97], the SICMA project [BKL+98], or the RIO project [SM98].

In our project we focus on the design of a multimedia storage network that manages a set of parallel disks as one virtual disk and that can be connected to several LANs and/or servers. Our design is not only resource efficient and able to support real-time requests, but it is also extremely fault-tolerant. This is mainly due to our new data placement approach. It keeps the data evenly distributed among the disks, even if new data or new disks are added. In other words, our data placement allows the disks to be perfectly "assimilated". The concept of assimilation is one of our main new contributions. Our data placement strategy furthermore

*Research supported by the DFG-Sonderforschungsbereich 376.

†Department of Mathematics and Computer Science, Paderborn University, Germany. Email: pebe@uni-paderborn.de

‡Department of Electrical Engineering, and Heinz Nixdorf Institute, Paderborn University, Germany. Email: brinkman@hni.uni-paderborn.de

§Department of Mathematics and Computer Science, and Heinz Nixdorf Institute, Paderborn University, Germany. Email: chrsch@uni-paderborn.de.

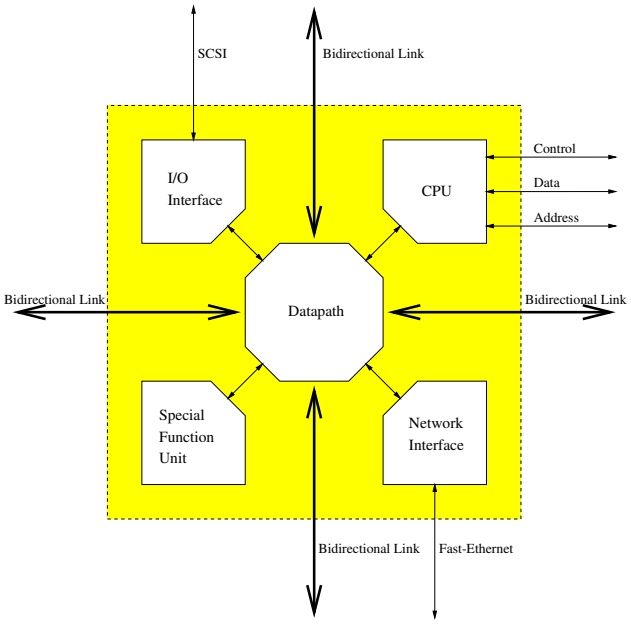


Figure 1: Structure of our active router

ensures a high fault-tolerance, since it prevents the loss of data with the help of local and global coding strategies and allows the fast recovery of lost redundant data in case of disk failures.

Section 2 gives an overview of our project. In Section 3 we focus on presenting a fault-tolerant, adaptive and resource-efficient data placement strategy. Furthermore, we will show as to why this strategy is an improvement over previously used strategies. In Section 4 we present simulation results for different data placement and load balancing strategies. The results confirm that our data placement and load balancing approach is superior to previous approaches.

2 The PRESTO Storage Network

The aim of the PRESTO project is to develop distributed algorithms and dedicated routing hardware for a very reliable storage network that manages a set of parallel disks in such a way that the network is ideally suited to support all kinds of multimedia applications.

The PRESTO storage network is aimed to be realized by a fixed connection network, based on the interconnection of hardware components of a single type. Every component can, in principle, be connected to a disk array (via SCSI) as well as to servers or LANs (via Ethernet). In order to organize a disk array, a RAID level may be used. RAID — which means "redundant array of independent (formerly, inexpensive) disks" — is a collection of standards, organized in so-called *levels*, that define how to place and access data in an array of disks.

In the following, we describe the principle characteristics of the PRESTO storage network. We only sketch the hardware concept and the communication strategy intended to be used in our system. Our main focus will lie on presenting efficient strategies for placing and accessing data.

2.1 The hardware concept

Our storage network is intended to be based on one single type of hardware component, called *active router*. Its structure is given in Figure 1.

This component has a SCSI interface for the connection of disks. It may store data blocks on these disks as defined by some RAID level. Furthermore, the active router has a Fast-Ethernet interface for the connection to the outside world and four dedicated links used for the exchange of information among the

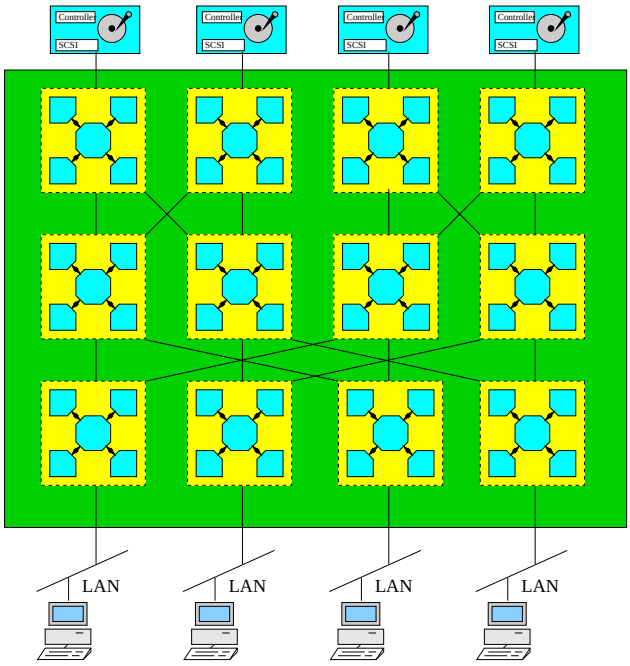


Figure 2: Active routers connected to a butterfly of depth 2.

active routers. These links are intended to support virtual channels for the exchange of data packets and control packets. A prototype of the active router already exists. It is based on an Ebsa 285 evaluation board with a SA-110 StrongARM processor. Implementations on this board are under way. Once the development of this prototype is completed, all components of our active router are planned to be integrated into one ASIC.

The great advantage of using only one type of component is that production costs are low and customers do not have to buy expensive, dedicated connection hardware when extending their storage system beyond some point. Especially the last item can be a serious problem when using storage systems that are based on buses. If the number of disks is too large to be supported by the given bus topology, it may be necessary to buy a completely new system that uses a bus with a higher bandwidth. Another advantage is the fact that, since our storage network has no dedicated routing center, it has no single device whose operation is decisive for the PRESTO network.

2.2 The communication strategy

In a first phase, we will concentrate on optimizing the performance of our routers when connected to a butterfly network. (The definition of a butterfly network can be found in a number of publications. Consult, e.g., [Lei92, S98].) The reason why we choose this type of network is that it has the minimum possible depth among all leveled networks of degree 4 that interconnect n servers with n disk arrays. See Figure 2 for a butterfly network of depth 3. Each disk symbol in this figure represents a disk array.

We recently found a routing protocol (see the tuned ghost packet protocol in [SV99]) which ensures that more than 99% of the available bandwidth of the butterfly network can be exploited if the number of packets that can be buffered at a dedicated link is a sufficiently large constant. This property is ensured, no matter how large the butterfly network is (that is, the size of the buffers does not depend on the network size, but only on the percentage of the available bandwidth up to which the protocol has to remain stable). Moreover, the protocol can easily be extended to tolerate network faults and to support multicasting, which is important for multimedia applications such as the common access to a multimedia object in a course or presentation.

3 Data management concepts

The PRESTO network is intended to manage its disks in such a way that it appears to the outside world as one single, giant hard disk that can support real-time requests. We assume the data space of this virtual disk (called *virtual data space* in the following) to consist of data blocks of uniform size, numbered consecutively from 0 to the maximally allowed block address. In order to be able to support real-time requests, suitable placement strategies for the blocks of the virtual data space have to be chosen. To simplify our ideas, we concentrate on the problem of distributing the virtual data space among a set of disk arrays, each equipped with 5 equally powerful disks. Our aim is to ensure that incoming requests can be evenly distributed among the disks in the long term and short term, no matter how their access pattern to the virtual data space looks like.

A well-known strategy to solve the long term load balancing problem is to use randomization, i.e., to place every block of the virtual data space at a randomly selected disk array. This ensures that every disk array is equally likely to be the target of a block request. Thus, in the long run, every disk array will have the same average share of the load. To store a random placement of the data, either large look-up tables or certain functions with a compact representation and pseudo-random properties (commonly called *hash functions*) can be used. We decided to use hash functions, since they can easily be stored at every routing element, so that every active router can compute the place of a data block locally. Look-up tables are not useful in our approach, since they would require to store an entry for every used data block (our storage network leaves the file management to the hosts connected to it, and therefore, does not know as to how the data blocks belong together). Hence it would be by far too expensive to store a look-up table in every active router. Pseudo-random hash functions, of course, have the drawback that they do not really distribute the blocks evenly among the disk arrays. However, when using large enough disks, the deviation from an even distribution is so small (below 1%) that this effect can be neglected.

A problem that has so far not been solved is the question of how to adapt the data placement to a changing number of disk arrays, which happens if disk arrays are added or removed. It is certainly not desirable to select a new hash function in this case that is completely unrelated to the previously used hash function, since this might require a complete remapping of the data blocks among the disks. Even if this problem could be solved, there may still be the problem that after the data remapping the disk arrays might not be equally likely to be accessed any more. Below we present a mapping strategy that, for every added or removed disk array, only require the remapping of a number of blocks that is in the order of the average number of data blocks stored at a disk array. Furthermore this strategy ensures that, in the long run, every disk is still equally likely to be accessed.

Apart from the problem of uniformly distributing the workload among the disk arrays in the long run, there is the problem of also ensuring short term load balancing, since otherwise it might not be possible to guarantee short delays for the real-time delivery of requests. Within every disk array, short term load balancing can be easily ensured, for instance, with the help of the RAID level 1, 3, or 5 placement strategies. To balance the load only within a disk array, however, does not suffice, since the distribution of requests among the disk arrays might be imbalanced. One approach to solve this problem (see, e.g., the RIO project) is to use *replication*, which means that several copies of some data blocks are stored at different disk arrays. In the following, we present a different approach. First, we present a strategy of how to place the data blocks and then we show how to use this placement strategy for short term load balancing.

3.1 The placement strategy

No matter of how large the storage network formed by our active routers is, we will assume as a default that it has a virtual data space of 2^{64} data blocks of uniform size (to be set by the system administrator), numbered from 0 to $2^{64} - 1$. However, only data blocks whose address has already been used are actually stored in the network (the rest is assumed to have a value of 0). That is, any address of the virtual space can be used to store blocks, as long as the number of used blocks within that space is below the maximum number of blocks that can be stored in the given storage network.

We intend to use the following coding and access strategies for the data blocks:

1. For data blocks that are much more frequently read than updated, we choose a simple parity strategy: Each block b is divided into k subblocks of equal size for some $k > 1$, named b_1 to b_k. Furthermore,

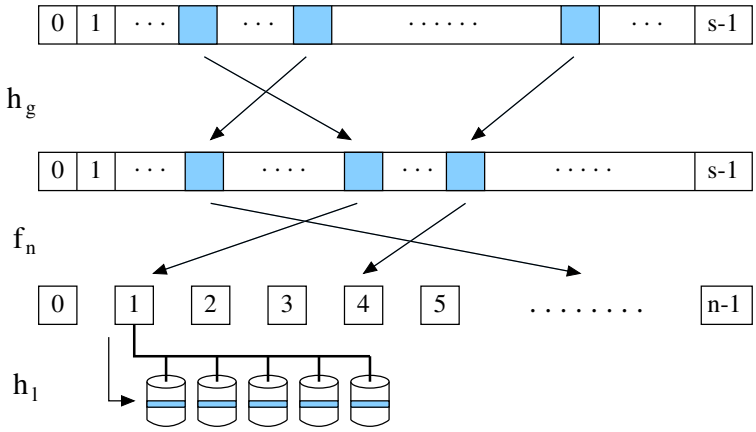


Figure 3: High-level structure of the placement strategy for n disk arrays.

a subblock b_{k+1} is created as the exlusive-or over b_1 to b_k. If a write request to block b is injected, all blocks b_1 to b_{k+1} are updated. Then it suffices for a read request to block b to access any k of the subblocks b_1 to b_{k+1} to be able to reconstruct the current value of b.

2. For data blocks that are updated more frequently, we choose the EVENODD strategy presented in [BBBM94]: Each block b is coded by $k+1$ subblocks $b_1, \ldots, b_{k+1}$ for some $k > 1$, each of size one $(k-1)$th of b. EVENODD allows b to be reconstructed from any $k-1$ of these subblocks. Thus it suffices to update only any k of the $k+1$ subblocks in case of a write request to ensure that, if any k of the $k+1$ subblocks are read in case of a read request, the current value of b can be reconstructed.

The strategies above ensure that, for the large majority of the requests, it suffices to access any k of the $k+1$ available subblocks of a block. As we will see later, this enables an efficient short-term load balancing of the requests. The coding strategies also have the advantage that, if the $k+1$ subblocks of each block are stored in different disk arrays, they ensure that, even if one of the disk arrays breaks down, we can still reconstruct every block from the surviving subblocks.

To simplify the presentation, we will only study in the following the real-time delivery of read requests for the coding strategy in (1) with $k = 4$. As we will see, this suffices to demonstrate that our strategy is superior to the replication strategy, concerning both redundancy and the ability to perform short-term load balancing at the disks and in the network.

For $k = 4$, the coding strategy in (1) has a redundancy of 25%. This is much more space-efficient than to simply replicate every data block. Another advantage of this strategy is that the congestion in the network caused by sending block data can be balanced more evenly (instead of one large stream of data, 4 smaller streams of data can be sent from different origins). Next we describe in detail how to place the subblocks.

Our placement strategy consists of 2 levels: a *global placement* (to distribute the data among the disk arrays) and a *local placement* (to place the data within a disk array). Figure 3 gives an overview of our strategies explained below.

Global placement. Let $s = 5 \cdot 2^{64}$ represent the number of subblocks that are to be placed and let n be the number of disk arrays. For the global placement, a suitable pseudo-random hash function, $h_g : [s] \to [s]$ with $[s] = \{0, \ldots, s-1\}$, is chosen to map every subblock address $b \in [s]$ to some address $h_g(b) \in [s]$. (We have chosen a function h_g that is based on the ISAAC random number generator [ISAAC], since it is very fast and appears to be a true random function even for very large domains.)

The disk array at which b has to be stored is selected with the help of $h_g(b)$ and an *assimilation function* $f_n : [s] \to [0, n]$. Given n disk arrays numbered from 0 to $n-1$, b is stored at disk array $\lfloor f_n(h_g(b)) \rfloor$. One of our major new contributions in this paper is to include f_n in the global placement scheme. f_n is selected in such a way that data can be efficiently remapped in case of a new disk array or the failure of a disk array.

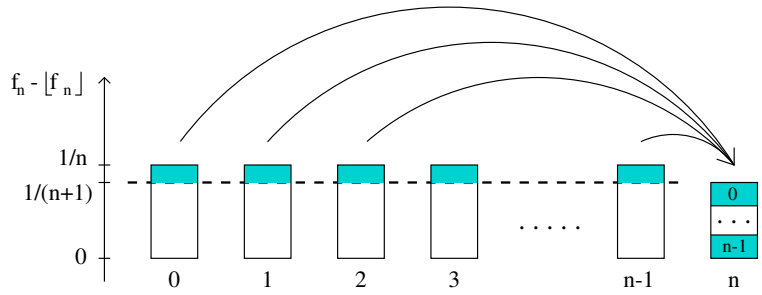


Figure 4: Remapping of data from n to $n+1$ disk arrays.

f_n is defined recursively. Let us start with $n = 1$, i. e. there is only one disk array. In this case we simple choose f_n to be $f_n(b) = b/s$. Now let us define how to get from f_n to f_{n+1}, that is, how to remap the data blocks stored at n disk arrays if another disk array is added. Intuitively, f_{n+1} results from f_n by taking away a portion of $1/(n \cdot (n+1))$ of the virtual data space from each of the n existing disk arrays and transferring it to the new disk array, as seen in Figure 4. Formally, if $f_n(b) - \lfloor f_n(b) \rfloor \geq 1/(n+1)$ then $f_{n+1}(b) = n + 1/(n+1) \cdot (1 - (\lfloor f_n(b) \rfloor + 1)/n) + [f_n(b) - \lfloor f_n(b) \rfloor - 1/(n+1)]$, otherwise $f_{n+1}(b) = f_n(b)$.

We show that with this definition of f_n the position of any address in $[s]$ changes at most $\lfloor \log n \rfloor + 1$ times from f_1 to f_n. Since it is easy to find the next time step a block address is moved (given a position $f_k(b)$ for block b, the first $\ell > k$ for which $f_\ell(b) \neq f_k(b)$ is equal to $\lceil 1/(f_n(b) - \lfloor f_n(b) \rfloor) - 1 \rceil$), the time complexity of computing the actual position of a virtual space address is $O(\log n)$, which is sufficiently small to be applicable in our storage network.

Lemma 3.1 *The position of any address in $[s]$ changes at most $\lfloor \log n \rfloor + 1$ times from f_1 to f_n.*

Proof. Consider any subblock b with $f_n(b) = m_0 + \epsilon_0$ for some $m_0 \in [n]$ and $\epsilon_0 \in [0, 1/n)$. The first time b is replaced, b will be moved to some disk array with number $m_1 = \lceil (1/\epsilon_0) - 1 \rceil \geq n$. In this case it holds that $f_{m_1}(b) = m_1 + \epsilon_1$ for some $\epsilon_1 \leq (m_1 - m_0)/(m_1 \cdot (m_1 + 1))$. Hence, the second time b is replaced, it will be moved to the disk array with number $m_2 \geq \lceil m_1 \cdot (m_1 + 1)/(m_1 - m_0) - 1 \rceil$. ¿From this it is fairly obvious that, for all choices of m_0 and ϵ_0, $m_2 \geq 4m_0$. This proves the lemma. □

¿From the recursive construction of f_n it is clear that only a portion of $1/(n+1)$ of the virtual data space has to be remapped when adding a new disk array to n disk arrays, and that this portion is taken evenly from all disk arrays. This is optimal for any remapping process that ensures an even distribution of the virtual address space among the disk arrays. Hence our placement strategy allows us to use a very efficient assimilation of a new disk array. It can also efficiently be used in case of a failure of a disk array: Assume that we have n disk arrays, and one fails. In this case, we simply let the working disk array with highest number take over the work of the malfunctioning disk array and remap the subblocks as prescribed when moving from f_n back to f_{n-1}, i.e. we re-assimilate the remaining disk arrays. (The way in which the subblocks are found that have to be remapped in case of an added or removed disk array is explained below.) Similar to these two primitives (remapping in case of a new or failed disk array), even more complex situations such as the simultaneous failure or disconnection of several disk arrays, or even the assimilation of two storage networks into one storage network (e. g. by combining their virtual data spaces into one virtual data space) can be handled efficiently. (However, the simultaneous breakdown of several disk arrays might result in the loss of data.)

Local placement. For the local placement of the data blocks, we assume that the active router has a sufficiently large memory to store a look-up table with entries for all subblocks stored in the disk array attached to the router. This assumption is realistic since, for a reasonable exploitation of the bandwidth of the disks, the subblocks have to be of size in the order of several 100 KB, which also ensures that the look-up table is not too large. Every entry stores the following information for one data subblock b: its number b, the position $p_b \in [d]$ where the subblock can be found in the disk array, its $(f_n - \lfloor f_n \rfloor)$ value and the numbers of the 4 disk arrays that store the other subblocks belonging to the same data block as b. The f_n values

are used for the remapping of datat in case of an additional disk array, and the disk array numbers are used for the remapping in case of a removed or malfunctioning disk array. They enable every active router to independently find those subblocks that need remapping by simply scanning its look-up table.

3.2 Short term load balancing

Our short term load balancing strategy works as follows. For every newly injected read request of a data block b, 5 control packets are generated and sent to the 5 disk arrays in which b's subblocks are to be found. Upon receipt of a control packet, every disk array sends a control packet back to the origin, containing the number of the requests that are waiting for disk access in its queue. After control packets have been received from all sites of b's subblocks by the node at which b was injected, the node sends subblock requests to those 4 disk arrays that reported the smallest number of waiting requests. (In order to break ties, if several disk arrays report the same number of requests, a random selection will be done.) This strategy is called *minimum game*. Such games have already been analyzed by several authors [KLM93, ABKU94, Mit96a, Mit96b, ABS98]. Not only in theory, but also in practice it did turn out that the minimum game is highly successful in balancing the load [Kor97, SM98]. The novelty of our approach is the use of a 4 out of 5 selection together with the parity strategy, instead of using a 1 out of 2 selection together with a replication strategy, as it was done in comparable storage network projects or products.

3.3 Related work

Most work on multimedia storage systems has concentrated on video and audio playback which has highly predictable access patterns. Once a video playback stream is started, data is sequentially read from the storage system at the playout rate. This predictability is exploited by many data placement and scheduling strategies that carefully lay out data on disks in order to achieve a reasonable load balance and real-time performance. See [CLOT96] for an overview and [CLOT96], [WLDH96], and [VRG95] for some examples. The most wide-spread scheme is data striping [BMGJ94, CPK95]. There are two basic striping techniques: either striping each data block over all disks or placing the blocks of the data space in a round robin fashion in the disk subsystems (usually RAID) of a storage network, one block per subsystem. Striping has been used, for instance, in the MARS project to design a high performance, large scale multimedia server [BPC94, BP97].

The drawback of using striping techniques, however, is the fact that they are either not scalable or perform poorly for non-predictable access patterns (as it is the case with interactive video on demand applications). To handle non-predictive access patterns, other projects such as SICMA and RIO use random placement strategies.

The SICMA project was set up to design a scalable data server for an interactive access to various forms of media and to demonstrate its efficiency by applying it to a relevant application, the "Virtual Museum". The server uses random placement without any replication in order to be resource efficient and to ensure an even distribution of requests among the disks, even for non-predictive access patterns. See [BKL+98] for a description of the project. Further publications of the project are [BRL96, RBL97].

To ensure the real-time delivery of requests even under heavy load, a partial replication of the data blocks is used in the RIO project to distribute the data requests more evenly among the disks [BMW96, SMB98, SM98, FSM98]. In [SMB98] it is shown that, even for predictable workloads such as video playback, the RIO system has a performance that is equivalent to conventional striping strategies. Fault-tolerance is achieved by using disk arrays of RAID level 5.

The use of replication for load balancing has the drawback of being fairly space-consuming, since the data is simply mirrored. A much more space-efficient technique is to use a parity strategy as was suggested above. Furthermore, none of the projects above can compensate the simultaneous failure of more than one disk in a storage subsystem, and no efficient data remapping strategies in case of new or removed storage subsystems have been investigated. The PRESTO network solves these problems with the help of the parity strategy and a global placement strategy that is based on a new assimilation technique.

4 Our Simulations

In this section we present simulation results that confirm that our short term load balancing and placement strategy is superior to previous strategies. First, we define our simulation model.

4.1 The simulation model and notations

In the following we assume that we have a storage network that connects 100 disk arrays to 100 servers. Every disk array consists of 5 disks. One of these disks is assumed to be a parity disk. All disks are of the same type. We assume the disk arrays to work in a synchronized way. One *round* is defined as the time needed by a disk to process a read request to a subblock, called *subrequest* in the following.(A round is intended to represent an upper bound of the time a disk needs to read a subblock, to be sure that deadlines do not fail due to variations of disk access times.) We assume that the subrequests can be distributed among the disks in such a way that in every round all disks of a disk array, apart from the parity disk, are able to process subrequests in parallel. That is, every disk array is able to process four subrequests in each round. In every round, the servers inject some fixed number of read requests to data blocks. This number is called *load* in the following. We assume the time all 5 control packets of a newly injected request r require to reach their disk arrays to be fixed. Furthermore, the number of rounds needed from this time point till the time when all 4 selected subrequests of r are placed in their respective disk queues is assumed to be fixed. This number is called *latency* in the following. To model the real-time delivery of requests, we assume every request to be equipped with a *deadline* d, limiting the number of rounds its 4 subrequests are allowed to wait in their disk queues for being processed. Every disk array processes the subrequests stored in its queue according to the EDF (earliest deadline first) rule. If one of the 4 subrequests fails to be processed within d rounds, it is removed from the disk queue and the corresponding request is declared as *cancelled*. The *waiting time* of a successful request is defined as the maximum number of rounds one of its 4 selected subrequests spent waiting in its disk queue.

4.2 The simulation results

We first show that our short term load balancing strategy is superior to strategies that are based on partial replication. A set of data blocks is said to be *partially replicated with redundancy* r if r% of the data blocks have two copies, each stored in a different disk array, and the rest only has one copy, stored entirely in one disk array. Data blocks that have two copies are said to be *replicated*. For the replicated data blocks, essentially the same selection strategy as described in Section 3.2 is performed: For every request to a replicated data block, two control packets are generated that are sent to the two alternative disk arrays to report the current number of requests stored in their disk queues. The request is sent to the disk array that reported the lower number of requests in its queue. To be able to compare the replication strategy with our parity strategy, we assume in the case of partial replication that in each round a disk array is able to process one request. (At this point, we are using a simplified disk model, neglecting the influence of the seek time of the disks. This discriminates the partial replication scheme, which is working with disk blocks that are four times larger. Knowing the disk parameters block size, seek time, and transfer rate, it is easy to account for these missproportions in our simulation results.)

Figure 5 shows the maximum load a partial replication strategy can sustain for different deadlines if the percentage of the requests that is cancelled is required to be below 10^{-6}. The latency is set to 3.

As can be seen, if the redundancy of the partial replication strategy is equal to the redundancy of our global partity strategy, namely 25%, the deadline has to be set above 20 to handle a load of about 90. In contrast, we observed (indicated by the "*" at position $(25, 89)$ in Figure 5) that even if the parity disks in the disk arrays are not used for the processing of subblocks, already a deadline of 6 allows our balancing strategy to handle a load of about 89 together with a fraction of cancelled requests that is below 10^{-6}. If the deadline is set to 6 for the partial replication scheme, it performs very poorly. Furthermore, if the deadline for our strategy is set to 11, it can handle a load of about 99 (indicated by the "*" at position $(25, 99)$), whereas for a deadline of 11 the partial replication scheme can only handle a load of about 65. If we allow the parity disks to be used in our load balancing strategy (i.e., we fully exploit the redundancy of about 56%), allowing to answer 5 instead 4 subrequests in parallel, then we observed that a deadline of 6 suffices

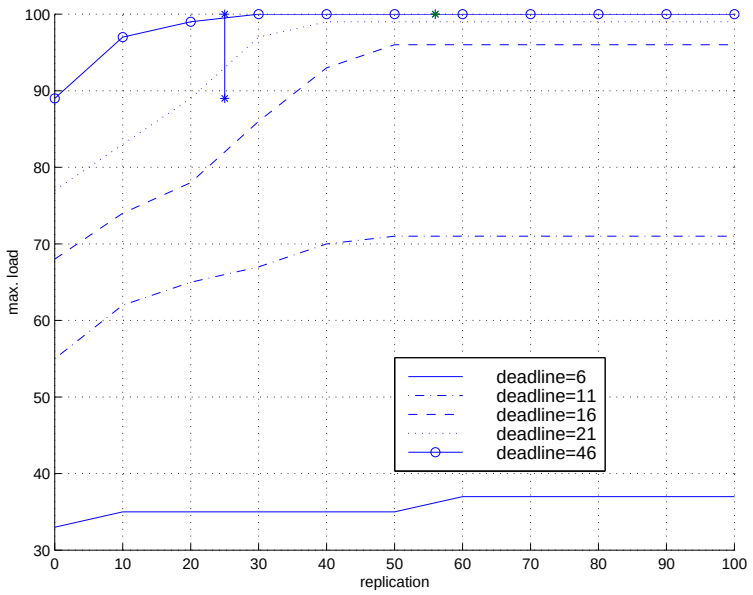


Figure 5: The maximum load a partial replication scheme can handle if the fraction of the cancelled requests is required to be below 10^{-6}.

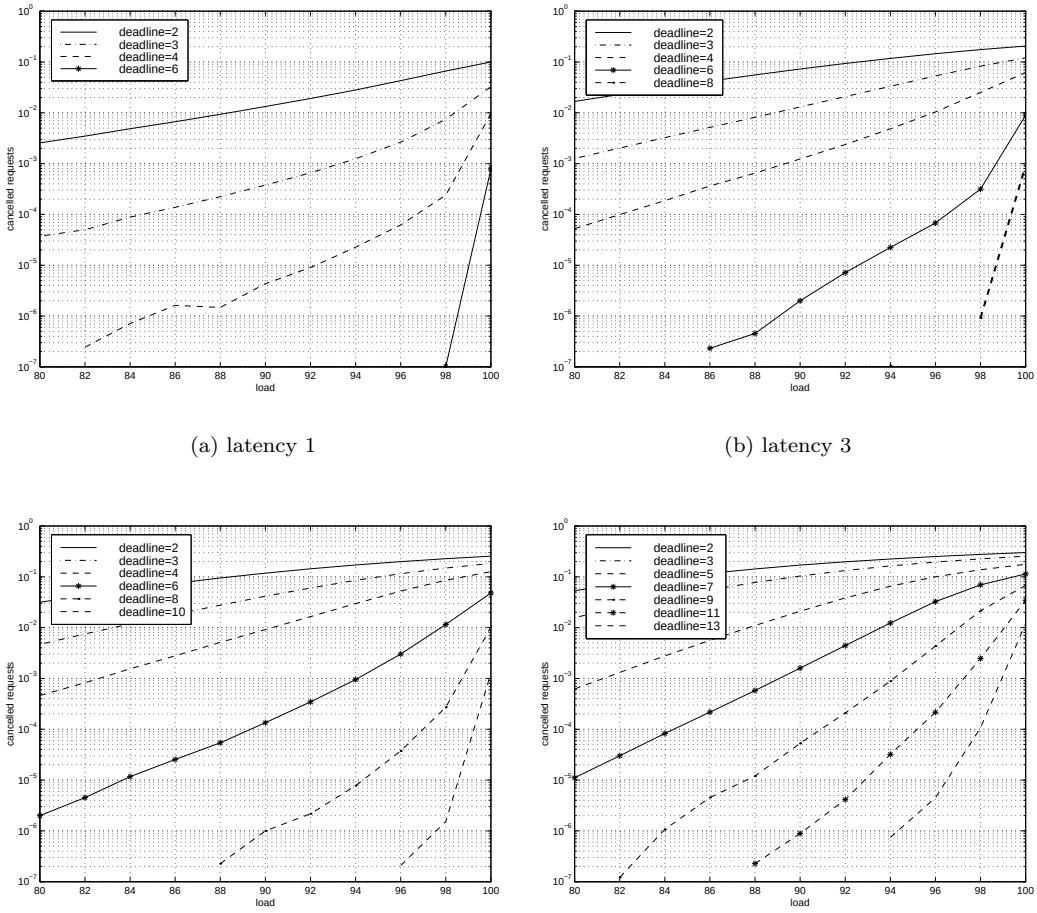
to handle even a load of 100 (indicated by the "*" at position $(56, 100)$).

The following figures present detailed simulation results about the performance of our load balancing strategy for a load of up to 100 requests per round and different deadlines and latencies. Note that, in case of a load of 100, the system would not be stable without deadlines (if the partity disks are not used). The deadlines prohibit that the number of subrequests stored in the disk queues grows unboundedly with time. Each of the figures represents the average over 10.000.000 injections of requests.

Figure 6 depicts the fraction of requests that are cancelled when using the parity strategy with different latencies and deadlines. As can be seen, in case of a latency of 1 already a deadline of 6 suffices to ensure a fraction of at most 10^{-6} cancelled requests for a load of up to 98. We observed that, for a latency of 3, resp. 5 or 10, a deadline of 8, resp. 10 or 15, has to be chosen to achieve a similar performance. This implies that the deadline necessary to ensure a fraction of at most 10^{-6} cancelled requests for the same maximum load might depend linearly on the latency. An explanation for this is that the larger the latency the more outdated is the information used by a request to decide which 4 of its 5 subblocks to access. This causes the distribution of subblock accesses among the disk arrays to get more imbalanced with an increasing latency. The degree of imbalance due to outdated information can be at most linear in the latency since this is an upper bound on the expected number of requests injected during latency many rounds that request subblocks in some fixed disk array.

5 Conclusions

In this paper, we presented the basic concepts for the design of the PRESTO multimedia storage network. We demonstrated that our data placement and load balancing strategies are superior to previously used strategies. In our data placement strategy, we introduced a new concept, called assimilation: each time a disk array is added or deleted, data is remapped in such a way that afterwards each disk array is again equally likely to be used by requests. As a next step, we intend to extend this assimilation approach also for an optimal exploitation of the available bandwidth of the given network: Each time, an active router is removed or added to the network, (close to) optimal paths are to be calculated so that the network load caused by data and control packets is as balanced as possible.



(a) latency 1

(b) latency 3

(c) latency 5

(d) latency 10

Figure 6: Fraction of cancelled requests for different latencies and deadlines.

References

[ABKU94] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. In *Proc. of the 26th Symposium on Theory of Computing (STOC)*, 1994.

[ABS98] M. Adler, P. Berenbrink, and K. Schröder. Analyzing an infinite parallel job allocation process. In *Proc. of the 6th European Symposium on Algorithms (ESA)*, pages 417–428, 1998.

[BBBM94] M. Blaum, J. Brady, J. Bruck, J. Menon. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. In *Proc. of the International Symposium on Computer Architecture*, pages 245-254, 1994.

[BKL+98] C. Brandt, G. Kyriakaki, W. Lamotte, R. Lüling ,Y. Maragoudakis, Y. Mavraganis K. Meyer, and N. Pappas. The SICMA multimedia server and virtual museum application. In *Proc. of the 3rd European Conference on Multimedia Applications, Services and Techniques*, pages 83-96, 1998.

[BMGJ94] S. Berson, R. Muntz, S. Ghandeharizadeh, and X. Ju. Staggered striping in multimedia information systems. In *SIGMOD 94*, pages 79-90, 1994.

[BMW96] S. Berson, R. Muntz, and W. Wong. Randomized Data Allocation For Real-time disks. Compcon 96, pages 286-290, 1996.

[BP97] M. Buddhikot and G. Parulkar. Efficient data layout, scheduling and playout control in MARS. In *Proc. of ACM Multimedia Systems*, pages 199-212, 1997.

[BPC94] M. Buddhikot, G. Parulkar, and J. Cox. Design of a large scale multimedia storage server. *Journal of Computer Networks and ISDN Systems*, pages 504-517, 1994.

[BRL96] P. Berenbrink, V. Rottmann and R. Lüling. A comparision of Data Layout schemes for multimedia servers. In *Proc. of the 1st Conference on Multimedia Applications, Services and Techniques*, pages 345-364, 1996.

[CLOT96] T. Chua, J. Li, B. Ooi, and K. Tan. Disk striping strategies for large video-on-demand servers. In *Proceedings of the 4th ACM Multimedia*, pages 297–306. 1996.

[CPK95] A.L. Chervenak, A.A. Patterson, and R.H. Katz. Choosing the best storage system for video service. In *Proc. of 3rd ACM Multimedia*, pages 109-119, 1995.

[FSM98] F. Fabbrocino, J.R. Santos, and R. Muntz. An implicitly scalable, fully interactive multimedia storage server. In *Proc. of the 2nd International Workshop on Distributed Interactive Simulation and Real Time Applications*, 1998.

[ISAAC] B. Jenkins. ISAAC: a fast cryptographic random number generator. http://ourworld.compuserve.com/homepages/bob_jenkins/isaacafa.htm

[KLM93] R. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. In *Proc. of the 24th ACM Symp. on Theory of Computing*, 1992.

[Kor97] J. Korst. Random duplicated assignment: An alternative to striping in video servers. In *Proc. of 5th ACM Multimedia*, pages 219–226, 1997.

[Lei92] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*. Morgan Kaufmann, San Mateo, CA, 1992.

[Mit96a] M. Mitzenmacher. Density dependent jump markov processes and applications to load balancing. In *Proc. of the 36th Symposium on Foundations of Computer Science*, pages 213–222, 1996.

[Mit96b] M. Mitzenmacher. The Power of Two Random Choices in Randomized Load Balancing. PhD thesis, Graduate Division of the University of California at Berkley, 1996.

[RBL97] V. Rottmann and P. Berenbrink and R. Lüling. A simple distributed scheduling policy for parallel interactive continuous media servers. *Journal of Parallel Computing*, pages 1757-1776, and Techniques, pages 345-364, 1997.

[S98] C. Scheideler. *Universal Routing Strategies for Interconnection Networks*. Springer Verlag, LNCS 1390, Heidelberg, 1998.

[SEK99] P. Sanders, S. Egner, J. Korst. Fast Concurrent Access to Parallel Disks. Technical Report MPI-I-1999-1-003, MPI, Saarbrücken, June 1999.

[SM98] J. Santos and R. Muntz. Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations. In *Proc. of the 6th ACM Multimedia*, pages 303–308, 1998.

[SMB98] J. Santos, R. Muntz, and S. Benson. A parallel disk storage system for real-time multimedia applications. *In Special Issue on Multimedia Computing Systems of the International Journal of Intelligent Systems*, 1998.

[SV99] C. Scheideler and B. Vöcking. From static to dynamic routing: Efficient transformations of store-and-forward protocols. In *Proc. of the 31th Symposium on Theory of Computing*, 1999.

[VRG95] H. Vin, S. Rao, and P. Goyal. Optimizing the placement of multimedia objects on disk arrays. In *Proc. of the International Conference on Multimedia Computing and Systems*, 1995.

[WLDH96] Y. Wang, J. Liu, D. Du, and J-H. Hsieh. Video file allocation over disk arrays for video-on-demand. In *Proc. of the 4th ACM Multimedia*, 1996.