Random redundant storage for video on demand

Random Redundant Storage for Video on Demand

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Aerts, Joep J.D.

Random redundant storage for video on demand / by Joep J.D. Aerts. -
Eindhoven: Technische Universiteit Eindhoven, 2003
Proefschrift
90-386-0602-8
NUR 919
Subject headings: combinatorial optimization / data storage / information retrieval
/ multimedia / integer programming
CR Subject Classification (1998) : H.2.4, H.3.2, H.3.3, G.1.6

Random Redundant Storage for Video on Demand

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr. R.A. van Santen, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op donderdag 16 januari 2003 om 16.00 uur

door

Joep Jozef David Aerts

geboren te Riel

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. E.H.L. Aarts
en
prof.dr. G.J. Woeginger

Copromotor: dr.ir. J.H.M. Korst

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).
IPA Dissertation Series 2003-01

Preface

During the last four years, my PhD project and my cycling ambitions continuously struggled for the highest priority in my life and even though some people might suspect differently, the main goal has always been to finish this thesis in time. Many people have helped me to realize this goal. Here, I would like to thank them.

First, I would like to thank my supervising team for their contributions to my research and their confidence in me reaching the final goal. I will never forget how much fun we had during our meetings. I thank Jan Korst for his invaluable support during the four years of research. His ideas, comments, and opinions have had a large influence on the results described in this thesis. I thank Wim Verhaegh for the daily support in the role of supervisor and cluster leader, and especially for his help in setting up the simulation experiments and reading every single letter I wrote. I thank Emile Aarts for convincing me to pursue a PhD, for providing me this great working environment with flexible working hours, and for his contribution to the final form of the thesis.

Then, I would like to thank Sebastian Egner, Michiel de Jong, Wil Michiels, Han La Poutré, Frits Spieksma, and Gerhard Woeginger for having had the opportunity to collaborate with them. I really enjoyed the discussions I had with each of them and it is really nice to realize that some of the results of these discussions ended up in joint papers or as contributions to this thesis. I thank Ramon Clout for his great help in preparing the thesis.

The research has been carried out at Philips Research in Eindhoven. I thank my colleagues of the Media Interaction Group for my pleasant stay. I owe special thanks to the roommates I had over the years in the students rooms: Anko, Antoine, Bob, Guido, Hettie, Johan, Nicolas, Paul, Peter, Ramon, and all the others. But above all, I would like to thank Wil and Marcelle for being my nearest colleagues for four years.

Finally, I would like to thank all persons who made my life outside work very enjoyable. First, I thank my team-mates and the team staff of cycling team “De Dommelstreek” and my skating friends of “E.s.s.v. Isis” and “IJsclub Tilburg”.

Working behind my desk was a lot easier if I could look forward to a good training in the evening or a race in the weekend. I owe many thanks to my parents, to Marjon, Ruud, and Nieke, and especially to Femke for the support and distraction they gave me. At last, I would like to thank Arjan (I really liked my holidays in Portugal) and Ruud (racing with my brother was even better than with my dearest team-mate) for supporting me during the defense of my thesis.

Contents

1

Introduction

1.1 Video on demand

Video on demand is an interesting alternative to video rental shops. Customers who have access to a video-on-demand system can order the movie they want to see without the need of leaving their homes. The customer can do this at any moment in time and he can watch the desired movie at full television quality and, preferably, with VCR-functionality, such as pause, resume, fast-forward, and rewind. Considering the decrease in cost of communication capacity, it is expected that video on demand will be implemented at a large scale in the very near future. Simpler services, such as pay-per-view, have already been implemented at a large scale, and video-on-demand systems can already be found in hotels and airplanes, where they make use of dedicated networks.

In video-on-demand systems video files are digitally stored in a so-called video server. Since videos are stored in a digital form, the playout of a video can be seen as a stream of bits to be transmitted from the video server to the customer who requested the video. When a customer requests a certain video, an admission control algorithm decides whether this request can be granted, and if so, the server sends the video over a communication network to the customer. As a video server

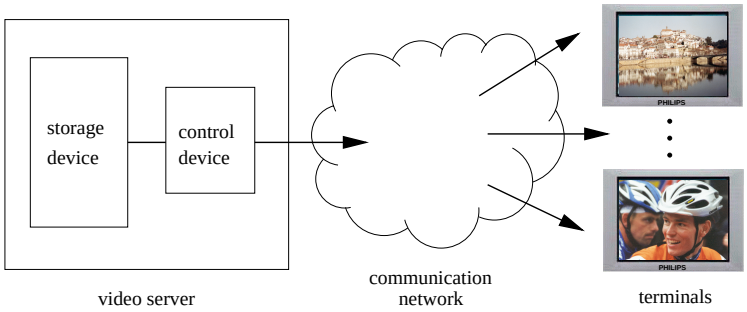


Figure 1.1. Model of a video-on-demand system.

is expected to serve a large number of customers we can see a video-on-demand chain as a client-server system. In the remainder of this thesis we use the following terminology. Clients request streams that have to be served by the video server. Within this client-server system we distinguish three parts as shown in Figure 1.1: the video server, in which the video data is stored and from which the system is controlled, a communication network, which connects the clients to the server, and the clients' terminals or set-top boxes.

In general, video data within a video server is stored on an array of hard disks. The transfer rate of a single hard disk is typically much larger than the bandwidth requirement for the playout of a single video, i.e. the maximum bit-rate of a video. This means that it would be very inefficient to reserve a single hard disk for each stream, apart from the fact that the data on that disk would not be available for other requests. Consequently, to use the full transfer capacity of the disk array, each disk should serve multiple streams, which means that the data of each stream becomes available in chunks. Accordingly, we store the video files on the hard disks in blocks, such that a video stream is formed by a sequence of data blocks and repeatedly a new block is retrieved for each client. A video is typically split up into thousands of data blocks.

A video-on-demand system offers continuous video streams to clients, ideally at the playout rate of the video, as in that case minimal buffer requirements are needed at the client's side. To enable a continuous stream from the server into the communication network, a buffer is implemented within the server for each stream. To deal with variable delays in the communication network, a second buffer may be implemented at the client's side. When a client is admitted service, he is assigned a part of the buffer space within the server. From this buffer the data can be sent into the communication network continuously at a variable bit-rate, which we assume to be equal to the playout rate of the video. The buffer is repeatedly refilled with data blocks from the disks. An internal network interconnects the disks and the

buffers. A buffer requests the next data block of the video file as soon as its filling is below a certain threshold.

The block requests have to be served by the disks of the disk array. These requests arrive at the disks over time. The disks have to retrieve the blocks in such a way that none of the blocks arrives too late at the buffers. This is what we call the disk scheduling problem. Due to unpredictable requests, as well as pause and resume actions of clients, this disk scheduling problem is on-line by nature. However, we translate it into a sequence of off-line problems by synchronizing the disks in the following way. We split up the time-axis into variable-length periods and in each period each disk retrieves a batch of blocks. While the disks are busy retrieving their batches of blocks, the new incoming requests are gathered. After all disks have finished, the next period can start. The new requests are distributed over the disks and each disk starts with its newly assigned batch.

In this thesis we focus on the server of a video-on-demand system. We do not discuss any further the communication network and the set-top boxes at the client's side. Within the server we focus on the storage and retrieval of video data. A storage strategy describes how the data blocks are distributed over the disks. Our main interest is in random redundant data storage strategies. In these strategies the data blocks are stored on randomly chosen disks and (some of) the data blocks are stored more than once. The randomness and redundancy are used to balance the load over the disks in order to fully exploit the transfer capacity of the disks. A retrieval algorithm describes how, in each period, the data blocks that are requested by the buffers are retrieved from the disks. For data redundant storage the main decision of such an algorithm is which disk to use for reading each block.

1.2 Informal problem statement

In this thesis we analyze the problem of designing storage and retrieval algorithms for the server of a video-on-demand system. In the design of storage and retrieval algorithms for a video server, a decision has to be made on the block size and the buffer strategy. An important requirement is that the buffers within the server never underflow or overflow. Besides satisfying this requirement, we want to find a storage and retrieval strategy that optimizes a certain criterion, such as minimum total cost or maximum number of streams that can be offered simultaneously for a given system configuration.

The input of the problem that we consider consists of some specifications of the system, such as the number of streams, the maximum bit-rate of a stream, the num-

ber of videos, or the number of disks in the disk array. The question is to design storage and retrieval algorithms for the video server that optimize certain criteria. Depending on the application the input parameters, specifications, and optimization criteria are different. For the choice which storage and retrieval strategy is most appropriate for a given setting, the disk efficiency of a storage strategy is important, which is defined as the fraction of the time that the disks spend on reading.

Two factors are of influence on the disk efficiency of a disk array. On the one hand there is the ability to distribute the workload over the disks, such that all disks work equally hard. This is what we call load balancing. On the other hand there is the efficiency of each single disk. Given that a disk has to retrieve a number of blocks, the question is how fast that can be done. We note that the size of the blocks influences the single disk efficiency. We readdress this issue in the next chapter. The disk efficiency of a storage strategy can be represented by the total time that it takes to retrieve a set of blocks. During the running of the system, we have to retrieve a set of blocks in each period, and the storage strategy that minimizes the period length performs best, regarding the disk efficiency. This means that the performance of a storage strategy can be measured by the ability to solve the following retrieval problem.

> **Retrieval problem**. Given are a storage strategy, a set of block requests, and for each block request the set of disks on which the block is stored. The question is to distribute the block requests over the disks such that the period length is minimized, which is defined as the time at which all disks have finished retrieving their blocks

The server should guarantee with high probability that the buffers do not underflow or overflow, even if in each period the number of blocks that have to be retrieved is as large as the maximum number of admissible streams. The latter is the case if the maximum number of clients is watching video and all clients consume at maximum bit-rate. This means that we take a worst-case point of view, in the sense that the system is fully loaded. In the next chapter we explain how we can realize the guarantee on the filling of the buffer, by linking the block size to the worst-case period length.

In conclusion, we analyze in this thesis the problem of designing an efficient video server, where we focus on the storage and retrieval strategies. Our main interest is the disk efficiency of random redundant data storage strategies. In the next chapter we give a formal description of the retrieval problems resulting from these strategies.

We analyze the storage and retrieval problems in a video server from a combinato-

rial optimization point of view. This means that we model the retrieval problems as combinatorial optimization problems and relate them to problems known in that domain. Our main goal is to design efficient algorithms for the retrieval problems of random redundant storage strategies. We use combinatorial optimization techniques to design these algorithms and to analyze the complexity of the problems and the performance of the algorithms.

1.3 Related work

In this section we describe relevant literature in the area of video-on-demand servers. These servers are also referred to as multimedia servers. We focus on literature that discusses storage and retrieval in these servers. We split this up into two parts: the first part discusses work on striping and the second part work on random redundant storage. Before that, we start with providing some pointers to papers that discuss specific parts within the design of a server that fall outside the scope of this thesis, such as the internal network and the buffer strategy.

In the design of a multimedia server a large number of choices have to be made. Gemmell, Vin, Kandlur, Rangan & Rowe [1995] and Shenoy, Goyal & Vin [1995] give a nice view of the issues that are involved in such a design. As stated, we distinguish within the server three parts. A disk array that stores the data, buffers or fast memory from which the data is sent out to the clients, and an internal network that interconnects the disks and the buffers. In the next chapter we discuss in detail the disk model, the internal network, and the buffer strategy that we use in the remainder of the thesis. Here, we give some references to prior work that discusses these issues in detail. For disk modeling we refer to Ruemmler & Wilkes [1994] and Oyang [1995]. Research on implementations of internal networks can be found in the work of Rehrmann, Monien, Lüling & Diekmann [1996] and Lüling & Cortés Goméz [1998] and in the references therein. Chang & Garcia-Molina [1997] analyze buffer requirements in a multimedia server for different disk scheduling policies. Dan, Dias, Mukherjee, Sitaram & Tewari [1995] investigate the trade-off between disk efficiency and buffer sizes and Korst, Pronk, Coumans, Van Doren & Aarts [1998] compare the performance of six buffering algorithms for multimedia servers.

An important choice in the design of the server is the block size. Two approaches to split up the video files or multimedia files can be found in the literature, being constant data length (CDL) blocks, where each block contains a constant number of bits, and constant time length (CTL) blocks, where the playout time of a block is constant. Vin, Rao & Goyal [1995] and Chang & Zakhor [1996] describe a

comparison between the two. Both papers conclude that CTL results in lower buffer requirements but increases the complexity of storage space management. Nerjes, Muth & Weikum [1997] give stochastic guarantees on period lengths when CTL blocks are used. However, most papers that discuss the implementation of a multimedia server use constant data length blocks, mainly because storing the data is easier. In this thesis we also assume CDL blocks.

A large number of papers have been published that discuss storage (or data placement) in multimedia servers. The remainder of this section deals with these papers and is split up into two parts. In the first part we discuss papers on striping strategies and in the second part papers on random redundant storage.

Striping. In the literature, most papers use disk striping strategies to distribute the video data over the disks. Several classes of striping strategies can be distinguished. In round-robin striping the consecutive blocks of a video file are stored in a round-robin fashion over the disks of the disk array. The result of this storage strategy is that a group of clients that is served by one disk in this period, is served by the next disk of the array in the next period. In full (or wide) striping each block is split up into as many subblocks as there are disks, and each subblock is stored on a different disk. This means that a request for a block results in a request for a subblock on each disk. A variant of full striping is narrow striping, where each block is striped over a subset of the disks. Several other implementations of striping have been proposed. We describe the main ideas of several papers that discuss striping techniques.

One of the first publications on disk striping is the work of Salem & Garcia-Molina [1986]. They refer to some earlier striping applications in Unix and Cray operating systems, but it seems that theirs is the first paper dedicated to disk striping. The main idea of the introduction of striping was increasing I/O bandwidth, to speed up processing. Also for that reason, Patterson, Gibson & Katz [1988] introduced the RAID (Redundant Arrays of Inexpensive Disks) technology, where increasing reliability was a second main objective. Later, the idea of striping was implemented in video servers to improve the load balance and thereby the efficiency of a disk array.

Chervanak, Patterson & Katz [1995] compare round-robin striping with the strategy of storing each video contiguously on disk. Round-Robin striping is a very popular strategy for multimedia servers, but it is less suited for variable bit-rate streams. Furthermore, this strategy results in large response times when the system is highly loaded. Chua, Li, Ooi & Tan [1996] propose a multi-disk implementation of the striping approach of Özden, Biliris, Rastogi & Silberschatz [1995] to

overcome the latter drawback. Berenbrink, Lüling & Rottmann [1996] use hashing functions and random placement to overcome the high response times at the cost of larger memory and processing requirements. To apply round-robin striping to variable bit-rate streams, Nerjes, Muth & Weikum [1997] use constant time length blocks. Berson, Ghandeharizadeh, Muntz & Ju [1994] discuss two striping strategies for streams with a higher bandwidth than the bandwidth of a single disk. In their simple striping approach each block is striped over a subset of the disks in a round-robin fashion, where the number of disks in a subset is determined by the ratio between the disk bandwidth and the required bandwidth. Staggered striping improves on this by being able to deal with heterogeneous streams.

Shenoy & Vin [1999] analyze two parameters of disk striping, being the choice of the size of the subblocks and the size of the subset of disks across which a stream is striped. They demonstrate in the paper that wide striping causes the number of admissible clients to increase sublinearly in the number of disks, and propose a narrow striping strategy where they partition the set of disks into subsets and stripe each stream over the disks within one subset. Korst [1997] states that narrow striping results in a load imbalance in case of variable bit rates. He shows that full striping does not scale well, in the sense that increasing the number of admissible clients results in a quadratic increase in the total buffer requirements. Other papers mention the same effect. Furthermore, full striping for large systems leads to very inefficient use of the disks, due to the high number of subblocks that have to be retrieved in each period.

Random redundant storage. Data redundant storage was first introduced to make systems less sensitive to disk failures; examples are the use of parity encoding in the work of Patterson, Gibson & Katz [1988] and chained declustering [Hsiao & DeWitt, 1990]. Parity encoding works as follows. For a set of blocks or subblocks a parity block is computed by taking the bitwise summation of the blocks. This parity block is stored on a disk that does not contain any of the blocks from which the parity block was constructed. In case of a disk failure the disk with the parity block can be used instead of the failing disk. In chained declustering each video is striped twice in a round-robin fashion over the disks of the disk array in such a way that two copies of the same block are stored on two subsequent disks. Papadopouli & Golubchik [1998] use the redundant data of the chained declustering storage strategy to improve disk efficiency. They describe a max-flow algorithm for load balancing. Merchant & Yu [1995] use more general duplicated striping techniques for multimedia servers. In their approach each data object is striped over the disks twice, where the striping strategy for each of the copies can differ. The redundant data is not only used for disk failures but also for performance improvements. Their

retrieval algorithm is based on shortest queue scheduling and the assigned requests are handled in FIFO (first in first out) order.

Berson, Muntz & Wong [1996] introduce random striping, where they split up each block into r subblocks and add an extra parity encoded subblock. These $r+1$ subblocks are randomly distributed over the disks and for the retrieval of a block any r of the $r+1$ subblocks are sufficient, such that the available parity blocks can be used for load balancing; they solve the resulting retrieval problem with a simple heuristic. Muntz, Santos & Berson [1998] and Tetzlaff & Flynn [1996] describe a system in which randomness as well as data redundancy is used for load balancing. Both use very simple on-line retrieval algorithms where requests are assigned to the disk with smallest queue. Tetzlaff and Flynn compare their results with coarse-grained striping and random single storage. Korst [1997] introduces a replication scheme in which each data block is stored on two randomly chosen disks, called random duplicated assignment (RDA). Korst analyzes the load balancing results of a number of retrieval algorithms, including heuristic algorithms as well as a max-flow based optimization algorithm, and compares their performance with full striping. Aerts, Korst & Egner [2000] extend on that paper. They proof a theorem that describes the maximum load, formulate an alternative max-flow graph, and discuss some special cases. Alemany & Thathachar [1997] independently introduce the same idea as Korst. They solve the retrieval problem with a matching approach. Sanders [2001] extends the RDA model to be able to take disk failures, splittable requests, variable size requests, and communication delays into account. Aerts, Korst & Verhaegh [2001] introduce a model in which a more accurate disk model is embedded, such that the multi-zone property of disks can be exploited to improve disk efficiency. Korst [1997] and Santos, Muntz & Ribeiro-Neto [2000] show that in case of variable bit-rates and less predictable streams, e.g. due to MPEG encoded video or VCR-functionality, random replicated storage strategies outperform the striping strategies. In case the bandwidth requirements of the server are the bottleneck instead of the storage requirements, this effect is even stronger.

In our analysis we assume that the data blocks are fetched periodically in batches. This means that all disks start with a new batch at the same time. Muntz, Santos & Berson [1998], Santos, Muntz & Ribeiro-Neto [2000] and Sanders [2000] analyze asynchronous retrieval strategies, where a disk can start with a new request as soon as it is idle. The first two papers use shortest queue scheduling in their real-time multimedia server. Sanders considers alternative asynchronous retrieval algorithms that outperform shortest queue scheduling. However, his analysis focuses on retrieving one request at a time, which means that seek optimization is not considered. The main reason to prefer asynchronous over synchronous re-

trieval is that by synchronizing the disks, a large fraction of the disks are idle at the end of each period. However, Aerts, Korst & Verhaegh [2002] show that the loss due to synchronization can be reduced to a very small fraction of the period length. Furthermore, when using synchronous retrieval it is easier to exploit seek optimization.

For periodic retrieval, probabilistic bounds can be derived on the load balancing performance. Several papers describe relevant probabilistic results in different settings such as Azar, Broder, Karlin & Upfal [1999] and Berenbrink, Czumaj, Steger & Vöcking [2000]. Azar et al. show that if n balls are placed one by one in m bins and for each ball two bins are available of which the least filled one is chosen, then the fullest bin contains $O(\log\log n/\log 2)$ balls with high probability. Berenbrink et al. give theoretical load balancing results for two on-line load balancing algorithms for throwing m balls into n bins, where $m \gg n$. For multimedia systems, Aerts, Korst & Egner [2000] and Sanders, Egner & Korst [2000] prove that random duplicated storage results in a good load balance with high probability.

1.4 Thesis contribution

The main interest of this thesis is the performance of random redundant data storage strategies. We describe two load balancing approaches, being block-based load balancing and time-based load balancing, each resulting in a different formulation of the retrieval problem. We link these retrieval problems to problems known in combinatorial optimization and in particular we show that they can be viewed as a special case of the multiprocessor scheduling problem [Pinedo, 1995].

In the block-based approach we model the period length by the maximum number of blocks that has to be retrieved from any of the disks. We relate the problem to the maximum density subgraph problem [Goldberg, 1984], where the objective is to find a subset of nodes that maximizes the ratio of the number of internal arcs to the number of nodes. Based on this relation we prove a theorem for the minimum load. Furthermore, we show that the problem can be modeled as a maximum flow graph [Ahuja, Magnanti & Orlin, 1989]. We develop a very fast parametric maximum flow algorithm [Gallo, Grigoriadis & Tarjan, 1989] for this retrieval problem.

In the time-based approach we model the period length more accurately. We take actual transfer times and switch times into account and we minimize in each period the maximum time that any of the disks is busy with retrieving its assigned block requests. This approach has several advantages over the block-based approach. First and most importantly, the disk efficiency improves, as we find a better load

balance and the approach enables the exploitation of the multi-zone character of disks [Aerts, Korst & Verhaegh, 2002]. The latter is quantified by a substantial increase in the fraction of blocks that is read from the fast outer zones. Furthermore, in contrast to the block-based approach the time-based approach can deal with heterogeneous streams or heterogeneous disks. With heterogeneous streams we mean that the data streams can have different maximum bit rates, e.g. due to different quality levels. A disk array with heterogeneous disks contains different disks, which means that the performance parameters of the disks, such as transfer rate and storage capacity, are not the same for each disk of the disk array.

We model the time-based retrieval problem as a mixed integer linear programming (MILP) problem [Nemhauser & Wolsey, 1989]. We prove that it is NP-complete in the strong sense [Garey & Johnson, 1979], but that it can be solved in pseudo-polynomial time if the number of machines is fixed. Based on the MILP model, we derive two approximation algorithms that start with the solution of the LP-relaxation and, to construct a feasible solution, perform a rounding and a matching procedure, respectively. Furthermore, we describe a new, very fast heuristic algorithm, based on list scheduling.

We analyze and compare the performance of the storage and retrieval strategies. We show with a probabilistic analysis [Rinnooy Kan, 1987] that the random redundant data storage strategies give good load balancing results with high probability. In addition, the simulation results show that the time-based approach doubles the number of requests in the fast outer zones compared to conventional strategies where zone location is not taken into account. To illustrate how to use the algorithms that are presented in this thesis in the design of efficient video-on-demand systems, we describe and analyze several applications of a server.

1.5 Thesis outline

The remainder of the thesis is organized as follows. In Chapter 2 we focus on the design of a video server. We explain how a disk and a server are modeled, we introduce several storage strategies, we formally define the retrieval problems, and show that the retrieval problems form a special class of multiprocessor scheduling problems. In Chapter 3 we dive into the block-based retrieval problem. We describe and analyze algorithms for the general problem as well as for some special cases. In Chapter 4 we discuss the time-based retrieval problem. We formulate the problem as an MILP problem and show that the problem is NP-complete in the strong sense. Furthermore, we analyze the complexity of some special cases, and describe and analyze algorithms. In Chapter 5 we analyze the performance

of random redundant data storage strategies and the corresponding retrieval algorithms with a probabilistic analysis as well as with simulations. In Chapter 6 we illustrate the effects of using random redundant storage strategies and the retrieval algorithms in the design of a video server. We describe several cases and analyze which storage and retrieval strategy fits best in a certain system setting according to a given optimization criterion. Finally, we summarize the results of this thesis and give some concluding remarks in Chapter 7.

2

Storage and Retrieval in a Video Server

A video server offers continuous streams of video data to multiple clients. In such a server we generally distinguish three parts, as shown in Figure 2.1: an array of hard disks to store the data, an internal network, and fast memory used for buffering.

As stated in the introduction, the video data is stored on the hard disks in blocks, which means that a requested video file is retrieved by repeatedly reading blocks from the disks. The blocks are then stored in the stream's buffer, from which the client can consume in a continuous way. The server should be able to serve a large number of clients simultaneously, thereby obeying some constraints, such as an upper bound on the response time, and optimizing some criteria, such as the cost per client.

In this chapter we describe in the first section the disk model that is used. In Section 2.2 we focus on the details of the complete video server and in Section 2.3 we introduce storage strategies. We formally define the block-based and time-based retrieval problems for redundant data storage strategies in Section 2.4. There, we also explain the relation between the retrieval problems and a special case of multiprocessor scheduling. We end this chapter with a discussion section.

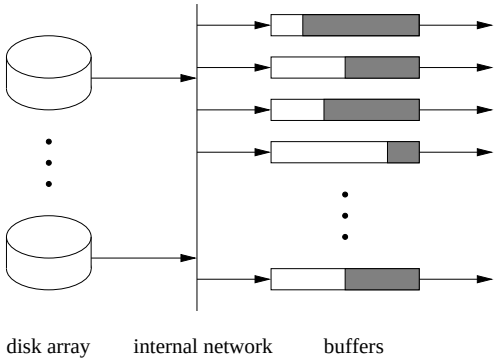


Figure 2.1. Model of a video server.

2.1 A disk model

The data that is provided by a video server is stored in blocks on the hard disks of the disk array. Unless stated otherwise, we assume that the disk array consists of a homogeneous set of hard disks and offers a homogeneous set of videos, where the latter means that the maximum bit-rate of each video is the same. In Chapter 4 we discuss the applicability of the time-based load balancing approach to heterogeneous settings.

We assume that the data blocks within the system are equally large, i.e. we use constant data length blocks. At the end of this section we discuss how this block size is determined. For now, we assume that we have blocks of a given size. The time a disk needs for the retrieval of a block is called the transfer time. The transfer time of a block depends on the location of the block on the disk. As a hard disk rotates at a constant angular velocity, and the outer tracks of a disk have a larger capacity than the inner tracks, a disk can read at a higher rate from the outer tracks than from the inner tracks. To exploit this, disks are split up in zones [Ruemmler & Wilkes, 1994]. Within a zone each track contains the same amount of data, which means that the transfer time is constant within a zone, but the transfer times for the subsequent zones decrease from inside to outside.

Between the retrieval of two successive blocks, a disk needs a certain amount of time, the so-called switch time, to move its read head from the end of the first block to the beginning of the next one. The efficiency of a disk largely depends on the switch overhead, i.e. the fraction of the time that is spent on switching. This means that retrieving the requests of a disk in an arbitrary order can result in inefficient disk usage. To decrease the switch overhead we retrieve the blocks from the disks in batches, such that the requests within one batch can be handled according to

their position on the disk. We assume that the disks use a SCAN-based sweep-strategy as presented by Coffman, Klimko & Ryan [1972], which means that a disk retrieves all blocks of a batch within one single sweep of the disk head. This sweep is either from the inside to the outside, or vice versa.

The total switch time of a batch equals the sum of the individual switch times between the retrievals of the blocks of the batch. Each individual switch time consists of a seek time, i.e. the time to move the disk head to the right track, and a rotational delay, i.e. the time that passes until the starting point of the block is under the disk head. In this thesis we use a simple worst-case estimation for rotational delay and seek time. We use the time of one full rotation r as an upper bound on the rotational delay. For the seek time we use a function that is linear in the number of tracks that have to be passed. In most disks this linear estimation is very accurate, as long as the number of tracks that have to be passed is not too small. For the seek time, the worst-case situation occurs when the requests are equidistantly distributed over the disk, and the disk head has to move from the innermost to the outermost track, or vice versa [Oyang, 1995]. We compute the distance between each two requests in this worst-case situation as the total number of tracks t divided by the number of requests i. Then, we can compute an upper bound on the total switch time with a function linear in the number of blocks of the sweep. This can be seen in the following way. A switch consists of a rotational delay r and a seek $a \cdot t/i + b$. Summing over the number of requests this gives the total switch time of a batch of i requests equal to

$$i \cdot (r + a \cdot t/i) + b) = i(r + b) + at = is + c \tag{2.1}$$

which is linear in the number of requests, with slope s and offset c. In Chapter 5 we define values for s and c that we use in the disk model for the simulation experiments. For an improved worst-case analysis of the performance of a hard disk we refer to Michiels, Korst & Aerts [2002]. However, for our analysis the simpler model is sufficient.

An important choice in the design of a video server is the size of the data blocks. To guarantee that the buffers within the server do not underflow, it is important that the blocks are large enough to offer video for the length of a worst-case period, which occurs if the maximum number of admissible clients is logged on to the system and the server has to retrieve exactly one block for each client. The buffers do not underflow if the playout time of a block is at least as large as the worst-case period length, either deterministically or statistically. Furthermore, in choosing the block size, a trade-off exists between disk efficiency and buffer size. If the data blocks are chosen larger, the disks can work more efficiently, as the switch overhead will be lower. On the other hand, the larger the blocks are, the larger the buffers need

to be. We discuss some of the trade-offs regarding the choice of the block size in Chapter 6. Before, we assume that the block size is fixed and that the system is configured in such a way that a block is large enough to offer video for the length of a worst-case period.

2.2 Video servers

Having explained the disk model, we continue in this section with the explanation of the working of the server. A client sends a request for a certain video to the server. An admission control algorithm within the server determines whether or not service is offered to this client. This admission control can be very easy, e.g. checking whether or not the number of running streams is less than the maximum number of streams that can be offered simultaneously. In case the server offers homogeneous streams, this admission control algorithm is sufficient, but in case of heterogeneous streams, due to, e.g. different quality levels, more sophisticated admission control is needed. We use in this thesis a straightforward admission control algorithm, unless stated otherwise.

When a client is admitted, he gets assigned a part of the buffer space within the server. This buffer space is usually implemented in fast solid-state memory. The client can consume data at a variable bit-rate from this buffer, but the rate is bounded by a maximum consumption rate, e.g. the maximum bit-rate of the video. The server has to guarantee that the buffer of the client never underflows or overflows, in order to guarantee continuous playout at the client's side. When the filling of a buffer is below a certain threshold, the buffer sends a request to the disk array for the next data block. The block is retrieved from the disks of the disk array and sent over the internal network to the buffer. We assume that the internal network is not a bottleneck, in the sense that there is always enough bandwidth available to transport the data from the disks to the buffers. The disks of the video server are synchronized, i.e. each disk gets a new batch of requests after all disks have finished their previous batch. This means that the period length equals the finishing time of the last disk.

Korst, Pronk & Coumans [1997] and Korst, Pronk, Coumans, Van Doren & Aarts [1998] discuss several buffer strategies for video servers. Throughout this thesis we use triple buffering. In this strategy each buffer can contain exactly three data blocks and it generates a request for the next data block in the upcoming period if the filling of the buffer is at most two complete blocks. Korst, Pronk & Coumans [1997] prove that this strategy guarantees that the buffers do not underflow or overflow, if the playout time of a block is at least as large as the worst case period

length. The proof is based on the fact that once a request for a new block is generated, this block always arrives within at most two worst-case period lengths, and the filling of the buffer is exactly enough to survive this amount of time.

As stated in the introduction the goal is to design a video server that guarantees that the buffers do not underflow or overflow, and next to that optimizes a certain criterion. Several criteria are possible such as the cost per client, the cost per video request, the response times, and the failure rate. Regarding the two cost criteria, we assume that the variable cost of the system consists of the cost of disks and buffers. All other costs are more or less independent of the storage and retrieval strategy. The response time is the time that passes between the request for a video and the start of the video at the client's side. When using triple buffering the client can start consuming from his buffer when the first block has arrived and the second block is requested. We do not consider the delay in the external network, hence the worst-case response time equals two times the worst case period length. The failure rate is the chance that a client does not get his data in time. As we do not consider the external network, we define the failure rate as the probability that a buffer underflow occurs. This is related to the probability that a period length exceeds the period length that is used to determine the block size in the design of the system.

2.3 Storage strategies

A storage strategy describes how the blocks of video data are stored on the disks. The choice of which storage strategy to use is an important choice in the design of the server, as it influences the optimization criteria introduced in the previous section. To optimize these criteria it is important that the available hardware within the server is used efficiently. Consider, for example, the naive storage strategy that stores each video contiguously on the disks. If a large fraction of the incoming requests requires the same video, in such a way that it is not possible to serve the streams in parallel, then this leads to an overload on one disk, whereas at the same time other disks are idle. For an efficient use of the disk array we must make sure that the work load is equally divided over the disks. This is what we call load balancing. The load balancing ability of a storage strategy is an important performance measure. Besides load balancing, it is also important that the individual disks are used efficiently. This means that switch overhead should be small and a large fraction of the blocks should be read from the outer zones.

In this section we introduce several storage strategies. As stated in Section 1.3 most papers propose disk striping strategies for distributing the data over the disks. We

start this section with discussing full striping. However, in this thesis we mainly focus on random redundant data storage strategies and use full striping for comparison. The first strategy of this kind that we introduce is a randomized version of coarse-grained striping and therefore we call it random striping. Afterwards we explain random multiplicated storage.

2.3.1 Striping

In full striping, also called wide striping, each block is split up into a number of subblocks, as many as the number of disks in the disk array. Each subblock is stored on its own disk such that a request for a block results in a request for a subblock on all disks. Figure 2.2 illustrates the storage strategy.

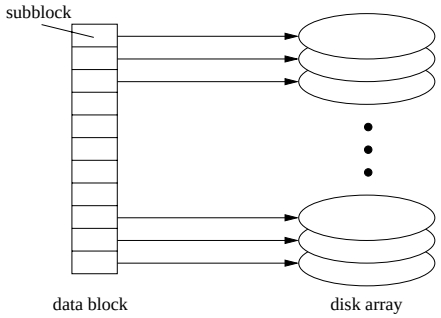


Figure 2.2. Full striping.

The retrieval strategy for full striping is straightforward, as each block request results in a subblock request on all disks, such that in each period each disk has to retrieve the same number of subblocks, i.e. the workload is equally spread over the disks. However, a drawback of full striping is that the number of disk accesses is as large as the number of requested blocks multiplied by the number of disks. This results in a large number of switches and consequently in a less efficient usage of the disks. Furthermore, this inefficiency also grows with the size of the system.

Full striping can deal with disk failures by introducing a parity disk. On this disk we store a parity subblock for each block [Shenoy & Vin, 2000], which is defined as the bitwise sum of the bits of the subblocks. In case of a disk failure the parity disk is used instead of the broken disk and the system performs in the same way as before. The server only has to do some basic computation to construct the requested blocks. The cost is one extra disk.

2.3.2 Random striping

In random striping we split up each request into r subblocks, where r is a parameter of this strategy. We compute for each block the parity subblock, again as a bitwise summation of the bits of the r subblocks. Now we have $r+1$ equal-sized subblocks and we store these blocks on $r+1$ different, randomly chosen disks. Figure 2.3 depicts this storage strategy for $r = 3$. A request for a block can be

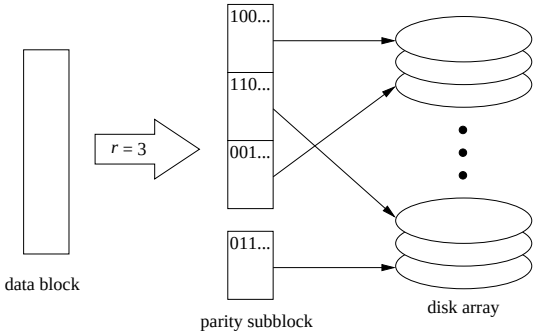


Figure 2.3. Random striping for $r = 3$.

served by retrieving the r original subblocks, or by retrieving $r-1$ of the original subblocks and the parity subblock. In the latter case the original block can easily be reconstructed. So, to serve a block request, we have to retrieve any r out of the $r+1$ subblocks. Compared to full striping we loose the guarantee that we get a perfect load balance. However, we have some freedom for each block, in choosing r out of the $r+1$ subblocks, and we can exploit this freedom to get a good load balance. This results in a retrieval problem, in which we have to decide for each block which disks to use for its retrieval such that the load is balanced. In the next section we introduce the retrieval problem in more detail.

If we assume constant a block size, the size of a subblock is determined by the value of the parameter r, and consequently the switch overhead depends on r as well; the smaller r is, the larger the subblocks are, the lower the switch overhead is. This means that for small values of r the disks can be used much more efficiently than in case of full striping. On the other hand, the smaller the value of r is, the larger the storage overhead is, e.g. for $r = 3$, we need 33% more storage space than strictly necessary. So, a trade-off between switch overhead and storage overhead has to be made. Depending on the ratio between the storage requirements and the transfer rate requirements of a system a suitable value for r can be determined. With respect to disk failures, no extra precautions are necessary, as the load of the failing disk can be equally spread over the remaining disks, due to the randomness.

However, the probability of a buffer underflow increases, as the expected period length increases.

2.3.3 Random multiplicated storage

In random multiplicated storage (RMS) strategies each data block is stored entirely on a number of randomly chosen disks. The multiplication factor can differ between various videos or even between the blocks of one video. An example of a random multiplication strategy is random duplicated storage (RDS) [Korst, 1997], where each data block is stored on two different, randomly chosen disks. Figure 2.4 illustrates RDS.

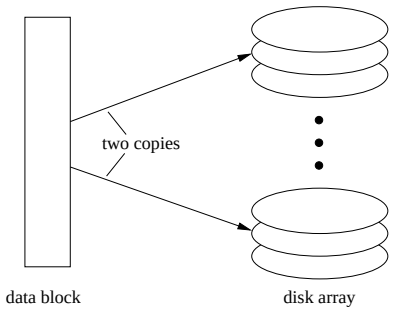


Figure 2.4. Random duplicate storage, a special case of random multiplicated storage.

As well as in the case of random striping, this storage strategy results in a storage overhead and a retrieval problem, so most observations made for random striping still hold. We can use the redundant data for load balancing and for surviving disk failures. However, this strategy results in a large storage overhead, but this is not a problem if the transfer capacity of the disks is the bottleneck instead of the storage capacity of the disks. As the storage capacity of disks grows considerably faster than the disk transfer capacity, this assumption becomes more and more realistic. In random multiplicated storage we do not split up the blocks into subblocks so we can use the transfer capacity of the disks very efficiently, as we can read full-size data blocks.

To decrease the storage overhead of random multiplicated storage and meanwhile keep the advantage of reading full-size blocks, it is possible to store only a fraction of the blocks twice. In partial duplication we store a fraction of the blocks twice and the remainder of the blocks once. The fraction is a parameter of the storage strategy and can be used for the trade-off between storage requirements and load

balancing performance. In case there are large popularity differences between the videos, it pays off to store the popular movies twice and the less popular movies once. The result is that in each period with a large probability the fraction of the requested blocks that is stored twice is larger than the fraction of total number of blocks that is stored twice. It is also possible to use an admission control algorithm that guarantees this, e.g. in the following way. In case a large number of clients is watching single stored videos, the server offers newly incoming clients only movies that are stored twice to choose from, in order to guarantee that in each period the number of requested blocks that are stored twice is large enough to enable good load balancing performance.

In the remainder of this thesis we explain the models and algorithms first for random multiplicated storage strategies. However, most models and algorithms work as well for the other redundant data storage strategies, random striping and partial duplication, and we point out how to extend the models and algorithms.

2.4 Retrieval problems

In the introduction we explained that the disks of the video server are synchronized and that the server works periodically. This means that for redundant data storage strategies in each period the following retrieval problem has to be solved. Given a set $J = \{1, \ldots, n\}$ of blocks that is stored on a set $M = \{1, \ldots, m\}$ of hard disks, select for each block the disk(s) from which it has to be retrieved such that the load of the disks is balanced.

2.4.1 Problem formulation

In the block-based retrieval approach we discard the differences in retrieval times, by assuming that the retrieval of a block takes a constant time for all blocks. The result of this constant time assumption is that the number of block requests assigned to each disk should be balanced. Minimizing the period length corresponds then to minimizing the maximum number of block requests assigned to one disk. This results in the following block-based retrieval problem.

Problem 1 [Block-based retrieval problem (BRP)]. Given are a set J of n blocks that have to be retrieved from a set M of m disks, and for each block $j \in J$ the set M_j of disks on which block j is stored. Select for each block j a disk from M_j, in such a way that the maximum number of blocks to be read from any disk is minimized.

The decision variant of BRP is defined as the question whether or not an assignment exists with a maximum load of at most K blocks per disk. The decision problem is only relevant for $K \geq \lceil \frac{n}{m} \rceil$, as otherwise no solution exists. □

In time-based load balancing we minimize the time on which the last disk finishes the retrieval of its assigned block requests. The completion time of a disk equals the sum of the retrieval times of the blocks plus the total switch time. As discussed before, we approximate the total switch time per disk by a function that is linear in the number i of block requests assigned to the disk, i.e. the switch time is set to $s \cdot i + c$, with the switch slope s and the switch offset c both at least zero.

The transfer time of a block depends on the zone of the disk where the block is stored [Ruemmler & Wilkes, 1994], where outer zones have a higher transfer rate than inner zones. The information of the zone location of blocks on disks is assumed to be available, so the retrieval times of each block on each disk are known beforehand. The decision of how to distribute the blocks over the zones is defined in the used storage strategy. We come back to this issue in Chapter 5 when we discuss implementation issues of a simulation.

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

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

- each block is fetched entirely, and
- the maximum completion time of the disks is minimized, where the completion time of a disk equals the sum of the total switch time and total transfer time.

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

2.4.2 Relation to multiprocessor scheduling

The discussed retrieval problems are related to scheduling [Pinedo, 1995] as defined within the field of combinatorial optimization. We can model the retrieval problems as scheduling problems by viewing the disks as machines and the requested blocks as jobs. The transfer time of block j then corresponds to the processing time p_j in the scheduling problem and the switch times of the retrieval problems correspond to set-up times. Using this correspondence we can call an assignment of the block requests to the disks a schedule.

As we consider an array of hard disks, the retrieval problems specifically relate to multiprocessor scheduling problems. Scheduling problems are often denoted in a three-field notation. We give a short introduction into this notation of scheduling problems and afterwards model BRP and TRP as such. For a more elaborate discussion of the three-field notation we refer to Pinedo [1995].

In the three-field notation the first field gives the machine environment, the second one describes the job characteristics, and the third one the optimization criterion. In the machine field a '1' indicates that we have a single machine environment. In the retrieval problems we have parallel machine environments, indicated by P or R, corresponding to identical machines and unrelated machines, respectively. The difference between P and R is that in case of P the processing time p_j of a job j is equal on all machines, whereas in case of R the processing time p_{ij} of job j also depends on the machine i. To indicate that we have a fixed number m of parallel identical machines we use Pm. For the second field we introduce four job characteristics that are necessary for the retrieval problems.

- Unit processing times are denoted by $p_j = 1$.
- Machine eligibility is denoted by M_j and means that only machines of subset M_j are available for job j.
- Set-up time, denoted by 'set-up', indicates that we need a certain amount of time to set up the machine before starting a new job. In the retrieval problems this corresponds to the switch slope.
- Preemption, denoted by 'pmtn', indicates that we allow job splitting. In the retrieval problem this means that we allow that a block is partially retrieved from different disks. Preemption in the retrieval problem is not exactly the same as preemption in the general scheduling literature. In scheduling it is not allowed to work with multiple machines on one job at the same time, whereas in the retrieval problems we allow that multiple disks retrieve parts of one block at the same time. We use 'pmtn*' to denote this variant of preemption.

Note that for problems with set-up times and without preemption, the set-up times can be added to the transfer times and do not change the nature of the problem.

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

Now we can formulate BRP and TRP for RMS as multiprocessor scheduling problems. In BRP we want to minimize the maximum number of block requests assigned to one disk. We model this by taking jobs with unit processing times in a parallel identical machine environment. Then, the number of jobs assigned to a machine becomes equivalent to the makespan. Furthermore, RMS makes that we have machine eligibility constraints, as for each job only a subset of the machines can be used. Concluding, in the three-field notation BRP can be denoted by $P\left|M_j,\ p_j = 1\right|C_{\max}$.

For TRP the machine environment is given by unrelated parallel machines, because the transfer time of a block depends on the zone in which it is stored. Again we have machine eligibility as a job characteristic. Furthermore, we have a set-up time for each job. This set-up time is constant, as we approximate the total switch time with a linear function. To enable partial retrieval we allow preemption. The optimization criterion is again the makespan, which is in this case the sum of the processing times and set-up times. Hence, in the three-field notation, TRP can be denoted by $R\left|M_j,\ \text{pmtn}^*,\ \text{set-up}\right|C_{\max}$.

2.5 Discussion

In this section we revisit some of the model choices. Hereby, we give more insight in the choices and trade-offs that arise in designing a video server. The choices and effects that are really important for the remainder of the thesis are discussed in the previous sections. The effects that we mention in this section are considered worth mentioning, but beyond the scope of the thesis to be discussed in detail.

Stream bandwidth. In the description of the video-on-demand system we distinguished three parts, being the server, the external network, and the clients. As we focus on the server we did not discuss the external network and the client side any further. However, as we describe system settings in Chapter 6 we want to give some extra comments. A client might need a second buffer, next to its buffer within the server, to deal with delay in the external network. This delay can be very unpredictable, e.g. when video on demand is implemented using the internet as external network, or very small and predictabel in case of a dedicated network, which might

be the case in a hotel. Furthermore, the read bandwidth out of the client's buffer within the server is bounded by a maximum rate. If we use MPEG encoded video, the peak rate of a video is much higher than the average rate, so using this peak rate as maximum bit-rate of a video would result in overdimensioning of the system. However, as we use blocks of data, bandwidth smoothing algorithms can be used to give a better estimation of the maximum bit-rate. For an overview of bandwidth smoothing algorithms we refer to Feng & Rexford [1999].

Internal network. We assume in this thesis that the internal network in the server is not a bottleneck. This means that the bandwidth of this network should be larger than the total bandwidth of the disk array, but next to this bandwidth requirement there is also a reachability requirement. The data that comes from the disks should be transported to the right buffer. For small servers this problem can be solved by using a large bus that interconnects each disk with each buffer. For larger systems the bandwidth and connectivity requirements ask for a more intelligent solution as described by Lüling & Cortés Goméz [1998] and their references. We do not consider this problem any further and assume that a sufficiently fast internal network can be constructed.

Prefetching. When introducing the storage strategies we presented partial duplication as a way to decrease the storage overhead at the cost of losing scheduling freedom. This loss can be compensated by using a different retrieval approach, which we call prefetching. In this approach each requested block needs to be retrieved in one of the t upcoming periods, where t is a parameter of this approach. The result is that we have scheduling freedom in two dimensions: in space (disks) and time. The cost of this approach is extra buffer space and a larger response time. The models and algorithms that are described in this thesis can be used to solve this alternative retrieval problem, but we do not discuss this approach any further. A somewhat similar strategy is discussed by Berenbrink, Riedel & Scheideler [1999] where they maximize the number of scheduled requests in case each incoming request has a deadline and a defined set of possible processors.

3

Block-Based Load Balancing

The first load balancing approach that we describe is block-based load balancing. In the block-based retrieval problem (BRP) we are given a number of blocks that has to be retrieved from a set of disks and for each block the set of disks is given on which the block is stored. The goal is to assign the blocks to the disks in such a way that the maximum number of blocks to be retrieved from any disk is minimized. We show in this chapter that BRP is solvable in polynomial time, by presenting several polynomial time algorithms.

This chapter is organized as follows. We discuss the modeling of BRP in Section 3.1, and thereby we relate BRP to problems known in the combinatorial optimization literature. In Section 3.2 we apply known maximum flow algorithms to BRP, being the Dinic-Karzanov algorithm, the preflow-push algorithm, and the parametric maximum flow algorithm. We show that the general time complexity results for these algorithms can be improved by exploiting the specific characteristics of the max-flow graph of BRP. In Section 3.3 we introduce a special case of BRP and describe a linear time algorithm for this case. We note that we first describe most models and strategies for random duplicate storage and explain afterwards how to extend them to other storage strategies.

3.1 BRP modeling

The block-based retrieval problem is related to a number of known combinatorial optimization problems. In this section we give a graph representation of BRP for duplicate storage and extract an integer linear programming (ILP) formulation from this graph. We describe how the formulation can be extended such that it is valid for other storage strategies. At the end of this section we relate BRP to the maximum density subgraph problem and explain how a maximum flow graph can be constructed.

3.1.1 ILP formulation

When each block is stored on exactly two disks, as is the case for RDS, we can model BRP with a so-called instance graph $G = (V,E)$, in which the set V of vertices represents the set of disks. An edge $\{i, j\} \in E$ between vertices i and j indicates that there are blocks for which the two copies are stored on disk i and disk j. For RDS the instance graph is the complete graph. In this graph we can represent an instance of BRP by putting on each edge $\{i, j\}$ a weight w_{ij}, that gives the number of blocks that has to be retrieved from either disk i or disk j. For ease of use we define $w_{ij} = 0$, if an edge $\{i, j\} \notin E$. Note that $\sum_{e \in E} w_e = n$. In Figure 3.1 we give an example of two nodes of an instance graph, representing two disks of BRP.

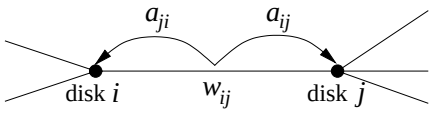


Figure 3.1. Example of two nodes of an instance graph of BRP for duplicate storage.

In this graph an assignment of block requests to disks corresponds to a division of the weight of each edge over its endpoints. We define $a_{ij} \in \mathbb{N}$ as the number of blocks of edge $\{i, j\}$ assigned to disk j and a_{ji} as the number of blocks assigned to disk i. Note that $w_{ij} = w_{ji} = a_{ij} + a_{ji}$. The load $l(i)$ of a disk i is given by the sum of the assigned weights of all incident edges, i.e. $l(i) = \sum_{\{i,j\} \in E} a_{ji}$. The load of the disk with maximum load is denoted by $l_{\max}$, i.e. $l_{\max} = \max_{j \in V} l(j)$.

With the above notation we can formulate the block-based retrieval problem for duplicate storage as an ILP. We call this special variant of BRP for RDS the edge weight partition problem.

Problem 3 [Edge weight partition problem]. Given is a graph $G = (V,E)$ with a nonnegative integer weight w_{ij} on each edge $\{i,j\} \in E$. Using the decision variables a_{ij} and a_{ji} for each $\{i,j\} \in E$ the problem is defined by the following ILP.

$$\begin{array}{lll} \min & l_{\max} & \\ \text{s.t.} & \sum\limits_{\{i,j\}\in E} a_{ji} \leq l_{\max} & \forall i \in V \\ & a_{ij} + a_{ji} = w_{ij} & \forall \{i,j\} \in E \\ & a_{ij} \in \mathbb{N} & \forall \{i,j\} \in E \end{array}$$

□

A solution of the edge weight partition problem can be transformed into a solution for BRP by specifying for each edge which blocks to retrieve from the two adjacent nodes.

The idea of load balancing is that we want to divide the load equally over the vertices of the instance graph, which means that we want to shift the load away from the parts of the graph where the edges have large weights. Given a subgraph $G' = (V',E')$, with $V' \subseteq V$ and $E' = \{\{i,j\} \in E | i,j \in V'\}$, we define the unavoidable load of G' as the sum of the weights of the edges of E' divided by the number of vertices of V', i.e. $\sum_{\{i,j\}\in E'} w_{ij}/|V'|$. This value is a lower bound on the value of an optimal load balance in G. The following theorem states that the optimal value is actually determined by the subgraph with maximum unavoidable load. We note that independently a similar theorem has been proven in another setting [Schoenmakers, 1995].

Theorem 3.1. *In case of duplicate storage we have*

$$l^*_{\max} = \max_{V'\subseteq V} \left\lceil \frac{1}{|V'|} \sum_{\{i,j\}\subset V'} w_{ij} \right\rceil . \tag{3.1}$$

Proof. It is easy to see that the right-hand side of (3.1) gives a lower bound on $l^*_{\max}$, since the total weight within a set V' has to be distributed over the vertices in V'. So we can prove equality by showing that we can construct a set $V^* \subseteq V$ such that

$$l^*_{\max} \leq \left\lceil \frac{1}{|V^*|} \sum_{\{i,j\}\subset V^*} w_{ij} \right\rceil . \tag{3.2}$$

Assume that we have an assignment for which the maximum load equals $l^*_{\max}$. Furthermore, without loss of generality, assume that the number of nodes with maximum load is minimal. We determine a node v^* with load $l^*_{\max}$. Initially, we set

$V^* = \{v^*\}$ and for this node v^* we determine neighbors $j \in V$ for which $a_{jv^*} > 0$. For such a neighbor j we know that $l(j) \geq l^*_{\max} - 1$, otherwise the load of v^* could have been decreased, without introducing another node with maximum load. This would contradict the assumption that the number of nodes with maximum load is minimal. We add these neighbors to V^* and continue recursively by adding for each $v \in V^*$ the neighbors j with $a_{jv} > 0$ to V^*. Also for these neighboring nodes j, it holds that $l(j) \geq l^*_{\max} - 1$, as otherwise we could find a path that could be used to decrease the load of a node with maximum load.

So, all nodes in V^* have a load of at least $l^*_{\max} - 1$ and node v^* has a load of $l^*_{\max}$. Following from the construction of V^*, no part of the loads of the elements of V^* can be assigned to elements outside of V^*. So the total weight on the edges within V^* is at least

$$\sum_{\{i,j\} \subset V^*} w_{ij} \geq (|V^*| - 1)(l^*_{\max} - 1) + l^*_{\max} = |V^*|(l^*_{\max} - 1) + 1,$$

hence

$$\left\lceil \frac{1}{|V^*|} \sum_{\{i,j\} \subset V^*} w_{ij} \right\rceil > l^*_{\max} - 1.$$

□

So the minimum maximum load is determined by the subset V' that maximizes $\left\lceil \frac{1}{|V'|} \sum_{\{i,j\} \subset V'} w_{ij} \right\rceil$. By transforming the graph of the edge weight partition problem into a multigraph by drawing w_{ij} edges between each pair i, j of vertices, the edge weight partition problem relates to the maximum density subgraph problem [Goldberg, 1984], which is the problem of finding a subgraph with maximum density in a multigraph.

Problem 4 [Maximum density subgraph problem]. Given is a multi-graph $G = (V, E)$. Find a subgraph $G' = (V', E')$ of G that maximizes $\frac{|E'|}{|V'|}$. □

Note that an optimal solution to the maximum density subgraph problem only gives a subset that gives a lower bound on the load. An extra step is needed to find a solution for BRP by distributing the blocks over the disks such that a load of $\left\lceil \frac{|E'|}{|V'|} \right\rceil$ is realized.

The graph representation and the integer linear programming formulation are simple formulations in case of duplicate storage. To illustrate that the ILP model can be extended to hold for other redundant storage strategies, we next give the ILP formulation for random striping with $r = 2$ and explain how Theorem 3.1 can be

adapted to hold for this case. For multiplicated storage and partial duplication the modifications are straightforward and therefore left out.

Recall that random striping with $r = 2$ means that each block is split up into two subblocks to which one parity subblock is added, and two out of these three subblocks have to be retrieved to reconstruct the original block. Given is a set of blocks that have to be retrieved from a set M of disks. Furthermore, for each combination of three disks $i, j, k \in M$, w_{ijk} gives the number of blocks for which the three subblocks are stored on these disks. We define for each such combination three decision variables a_{jki}, a_{kij}, and a_{ijk} that give the number of blocks of w_{ijk} assigned to disk i, disk j, and disk k, respectively. Then we can formulate an ILP as follows.

$$
\begin{array}{lll}
\min & l_{\max} & \\
\text{s.t.} & \sum_{j,k \in M} a_{jki} \leq l_{\max} & \forall i \in M \\
& a_{jki} + a_{kij} + a_{ijk} = 2w_{ijk} & \forall \{i,j,k\} \subset M \\
& a_{jki} \leq w_{ijk}, a_{kij} \leq w_{ijk}, a_{ijk} \leq w_{ijk} & \forall \{i,j,k\} \subset M \\
& a_{ijk} \in \mathbb{N} & \forall \{i,j,k\} \subset M
\end{array}
$$

By using this extended ILP formulation a theorem similar to Theorem 3.1 can be proven for random striping. The idea of unavoidable load within subsets of disks remains valid, if we redefine the unavoidable load of a subset V'. For the case $r = 2$ this gives

$$
2 \sum_{\{i,j,k\} \subseteq V'} w_{ijk} + \sum_{\{i,j\} \subseteq V', k \in V \setminus V'} w_{ijk}. \tag{3.3}
$$

3.1.2 Maximum flow formulation

Now we show that the decision variant of BRP can be formulated as a maximum flow problem. As the maximum flow problem is known to be solvable in polynomial time, this correspondence implies that BRP is solvable in polynomial time. We define a directed max-flow graph for random multiplicated storage as follows. The set of nodes consists of a source s, a sink t, a node for each disk, and a node for each requested block. The set of arcs consists of

- arcs with unit capacity from the source to each block node,
- arcs with unit capacity from each block node j to the disk nodes corresponding to the disks in M_j, and
- arcs with capacity K from each disk node to the sink, where K is the maximum allowed load.

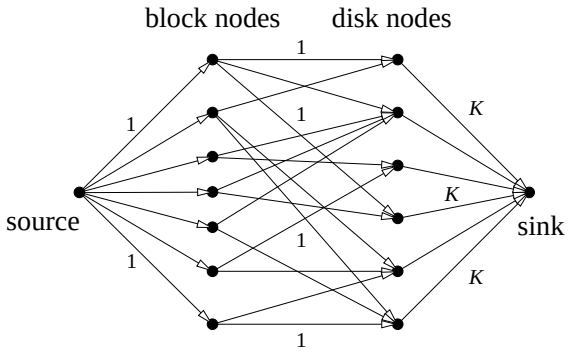


Figure 3.2. Example of a max-flow graph for the decision variant of BRP.

Figure 3.2 gives an example of such a max-flow graph.

We can solve the decision variant of BRP by finding a maximum flow in this graph. Recall that a network with integral capacities admits a maximum flow for which the flow over each edge is integral [Ahuja, Magnanti & Orlin, 1989]. If an integral maximum flow from source to sink saturates all the edges leaving the source, then this flow corresponds to a feasible assignment. This solution approach does not only solve the decision problem, but also gives an assignment in case of a positive answer, which can be derived from the flow over the arcs between the block nodes and the disk nodes.

An algorithm that performs a bisection search over the maximum allowed load K solves the optimization problem and shows that BRP can be solved in polynomial time. This proofs the following theorem.

Theorem 3.2. *BRP is solvable in polynomial time.* □

We can easily change the graph such that it holds for random striping. We increase the weights on the edges leaving the source to r, the number of subblocks in the random striping strategy. Furthermore, the number of edges leaving each block node equals $r+1$. One unit flow in this graph corresponds to a subblock.

3.2 Maximum flow algorithms for BRP

In this section we explain three maximum flow algorithms from literature, the Dinic-Karzanov algorithm [Dinic, 1970; Karzanov, 1974], the preflow-push algorithm [Goldberg & Tarjan, 1988], and the parametric maximum flow algorithm [Gallo, Grigoriadis & Tarjan, 1989]. We describe how the general time complexity results of these algorithms can be improved using the graph characteristics

of the max-flow graph of BRP. We start with the Dinic-Karzanov algorithm.

3.2.1 Dinic-Karzanov maximum flow algorithm

Consider the max-flow graph $G = (V,E)$ with capacity $c(e)$ on each arc $e \in E$ as defined in the previous section. The algorithm starts with an empty flow, and increases the flow step by step by sending additional flow over augmenting paths. In each stage the current flow f is increased by a flow g which is constructed in the following way. We follow the formulation of Papadimitriou & Steiglitz [1982].

(*i*) Compute augmenting capacities. We start with constructing an augmenting network $G(f)$ for the current flow f. The capacities in $G(f)$ are the augmenting capacities of the original network $G = (V,E)$ in which a flow f already exists. An arc (u,v) of the original graph G occurs in $G(f)$, if the arc is not saturated by f, i.e. $f(u,v) < c(u,v)$; the capacity in $G(f)$ then equals $c(u,v) - f(u,v)$. Furthermore, an arc $(u,v) \in E$ with $f(u,v) > 0$ results in the reverse arc (v,u) in $G(f)$; the capacity of (v,u) in $G(f)$ equals $f(u,v)$.

(*ii*) Construct the auxiliary network $A(f)$. Label the nodes in $N(f)$ such that the label of a node gives the shortest distance (in number of edges) from the source to that node. As we are looking for shortest augmenting paths we omit all nodes with a distance larger than or equal to the *s*-*t* distance, i.e. the distance label of the sink. Furthermore, for the same reason we omit arcs that are not directed from a node with label j to a node with label $j+1$. That leads to the auxiliary network $A(f)$.

(*iii*) Find a blocking flow g in the auxiliary network. First we note that we do not aim at finding a maximum flow in this step, but that we want to find a blocking flow, i.e. a flow that cannot be increased by a forward augmenting path. To find a blocking flow, we start with defining the throughput of each node as either the sum of the incoming arcs or the sum of the outgoing arcs, depending on which of the two is smaller. Then, we take the node with minimum throughput and push from this node an amount of flow, equal to the throughput, to the sink. This is done in a breadth-first manner, such that each node needs to be considered only once during a push procedure. As we take the minimal throughput in each step, it is guaranteed that each node can push out its incoming amount of flow. In a similar way the same amount of flow is pulled from the source. After the push and pull we remove the saturated arcs, update the throughput values, remove the nodes with throughput zero, and take again the node with minimum throughput for the next push and pull step. We continue until no path from source to sink exists, which means that we have constructed a blocking flow.

After each iteration of the above three steps, we add g to f and continue with the next iteration. The algorithm terminates when source and sink are disconnected in the auxiliary network.

For general graphs the Dinic-Karzanov algorithm finds a maximum flow in $O(|V|^3)$ time. We next show that for BRP this algorithm has a time complexity of $O(mn)$ for fixed K and leads to $O(\min\{n^2, m^2n, mn\log n\})$ for finding the optimal K. These statements hold in case the size of the sets M_j is bounded by a constant. We first state Dinic's lemma that states that the shortest augmenting path increases every iteration. For the proof we refer to Papadimitriou & Steiglitz [1982]. We need this result to prove the time complexity results for BRP.

Lemma 3.1. *In each stage the s-t distance in $A(f+g)$ is strictly greater than the s-t distance in $A(f)$.* □

Theorem 3.3. *The Dinic-Karzanov max-flow algorithm for the decision variant of BRP has a time complexity of $O(mn)$, in case $|M_j| = O(1)$ for all j.*

Proof. For the complexity of the algorithm we bound the number of stages of the algorithm and the time complexity of each stage. With respect to the number of stages, Lemma 3.1 states that in each stage the length of the shortest augmenting path increases. This means that the number of stages is bounded by the length of the longest path in the original max-flow graph, allowing reverse arcs in the path. The longest path alternates between disk nodes and block nodes and, as each disk node can be visited at most once, the length of the longest path is $O(m)$.

In each stage of the algorithm we find a blocking flow in the auxiliary network with respect to the current flow. We start with computing the augmenting capacities; this takes $O(|E|) = O(n)$ time, as the size of each set M_j is bounded by a constant. Constructing the auxiliary network $A(f)$ from $N(f)$ can also be done in $O(n)$ time, by doing the labeling in a breadth-first manner. When finding the blocking flow we know that the arcs with unit capacity are visited at most once, as they are saturated immediately. As $|M_j| = O(1)$, there are $O(n)$ of these arcs, and consequently $O(n)$ augmentations. Also the number of augmentations on the other arcs, i.e. the arcs from the disk nodes to the sink, can be bounded by n, as the maximum flow is at most n and these arcs never occur backwards in an augmenting path. This gives that the time complexity in each stage is $O(n)$.

Combined with the time bound on the number of stages, the overall time complexity of the max-flow algorithm for the decision variant of BRP is $O(mn)$. □

Theorem 3.4. *The Dinic-Karzanov algorithm solves the optimization variant of BRP in $O(\min\{mn\log n, m^2n, n^2\})$ time, in case $|M_j| = O(1)$ for all j.*

Proof. We show that each of the three components gives a bound on the complexity

of solving BRP.

1. A trivial lower and upper bound on the value of K is $\lceil n/m \rceil$ and n, respectively, such that a bisection search on the value of K solves BRP in $O(mn \log n)$ time.
2. For the second bound we show that for at most m different values of K a max-flow has to be solved. This can be seen in the following way. After solving the max-flow first for $K = \lceil n/m \rceil$, either we have a feasible solution, or at least one of the edges from the disk nodes to the sink is not saturated. Increasing the value of K on one of these edges does not improve the solution, such that we can continue with a new max-flow graph containing only a subset of the block and disk nodes. We construct the new value of K as follows. We add to the old value the number of blocks that are not yet assigned divided by the number of disks that had a load of K in the previous step and round this value up to the next integer. For this K, again we can conclude that either a solution is found or the number of saturated arcs from the disk nodes to the sink decreases. The number of saturated arcs from disk nodes to the sink decreases in each step such that we have at most m steps. This gives a total complexity of $O(m^2 n)$.
3. A third way to derive a complexity bound is by bounding the total number of times an auxiliary network is constructed and a blocking flow has to be found, without distinguishing between different values of K. The maximum flow at the end of the algorithm equals n and each blocking flow increases the total flow with at least 1, such that the total number of times a blocking flow is constructed is bounded by n. By starting with $K = \lceil n/m \rceil$ and updating K in the same way as above, the number of times an auxiliary network is constructed is $O(n)$, such that BRP can be solved in $O(n^2)$.

□

For practical situations the assumption that $|M_j| = O(1)$ is not a restriction, as the maximum multiplication factor in any relevant storage strategy is always bounded by a constant. Note that if $|M_j|$ would not be bounded by a constant, $|M_j|$ is at most m, such that the time complexity bounds in Theorems 3.3 and 3.4 grow at most with a factor m.

In case of duplicate storage, i.e. $|M_j| = 2$ for all blocks, an alternative graph formulation gives another time complexity bound. Korst [1997] describes a max-flow graph with m disk nodes and no block nodes, in which the maximum load of a given assignment can be decreased by finding a flow from disks with a high load to disks with a low load. Korst describes an algorithm that is linear in n for find-

ing a feasible starting assignment and solves the retrieval problem optimally with $O(\log n)$ max-flow computations, each of which can be done in $O(m^3)$ time. This gives a time complexity bound of $O(n + m^3 \log n)$. Based on the work of Korst, Low [2002] describes a tree-based algorithm that runs in $O(n^2 + mn)$ time. His algorithm can also be applied to random multiplicated storage.

3.2.2 Preflow-push maximum flow algorithm

In this section we explain the preflow-push algorithm [Goldberg & Tarjan, 1988] and analyze its time complexity for the BRP graph. The preflow-push algorithm solves the max-flow problem in $O(|V||E|\log(|V|^2/|E|)$ for general graphs, which equals $O(|V|^3)$ in case of dense graphs. We show that the algorithm solves the retrieval problem for a fixed K in $O(mn)$ time. In the next section we show that this complexity bound remains valid for the optimization variant by modeling BRP as a parametric max-flow problem.

The idea of the preflow push algorithm is to push the maximum amount of flow into the network, try to push as much flow as possible to the sink, and push the remaining flow back to the source. In contrast to the Dinic-Karzanov algorithm we do not require a feasible flow in each step of the algorithm. Instead, we relax the flow conservation constraint as follows. A flow is defined to be a preflow if the flow into each vertex is at least as large as the flow out of that vertex, except for the source. A preflow is a feasible flow if the inflow equals the outflow in all nodes, except for the source and sink. To control the pushing of flow, each node gets a height label and flow can only be pushed downhill. The algorithm has two basic operations, (*i*) push, to push flow from an overflowing vertex to a connected 'lower' vertex, and (*ii*) lift, to increase the height of an overflowing node to be able to push flow downhill in a next step.

Again we consider a directed graph $G = (V, E)$ with a source s, a sink t and capacities c on the edges. Let f be a preflow in G and c_f be the residual capacities according to f, i.e. for each $(u,v) \in E$ we get $c_f(u,v) = c(u,v) - f(u,v)$ and $c_f(v,u) = f(u,v)$. Furthermore, let $h : V \to \mathbb{N}$ be a height function, which is called feasible if $h(s) = |V|$, $h(t) = 0$, and $h(u) \leq h(v) + 1$ for every residual edge (u,v). We define the net flow into a node u as the excess of u, $e(u)$. Note that $e(u) \geq 0$ and that if preflow f is a feasible flow, then $e(u) = 0$ for all u. Now we can specify the two basic operations.

- **Push**. The procedure push can be applied to a directed edge (u,v), if (*i*) vertex u is overflowing, i.e. $e(u) > 0$, and (*ii*) $h(u) = h(v) + 1$. We push the maximum amount of flow, i.e. $\min\{e(u), c_f(u,v)\}$, over (u,v), and decrease

$e(u)$ and increase $e(v)$ with this amount. If the edge (u,v) is saturated after the push, it was a saturating push and the residual capacity $c_f(u,v)$ becomes zero.

- **Lift**. The procedure lift can be applied to a vertex u, if (*i*) u is overflowing and (*ii*) for all residual edges (u,v) we have $h(u) \leq h(v)$. We increase the height of u such that at least one of the residual edges can be used to push flow, i.e. $h(u) = 1 + \min\{h(v) | c_f(u,v) > 0\}$. Note that lift gives the vertex the maximum height that is allowed by the constraints on the height functions.

We initialize the max-flow graph in the preflow-push algorithm as follows. We set $h(s) = |V|$ and $h(u) = 0$ for all $u \in V \setminus \{s\}$. Furthermore, we push the maximum amount of flow into all edges connected to s, i.e. $f(s,u) = c(s,u)$ for all nodes $u \in V$ that are adjacent to s. All other edges become residual edges with residual capacity equal to the original capacity. Then we start executing push and lift operations until we can no longer apply a lift or push on any of the nodes. In the resulting graph the conservation of flow constraint holds in all nodes, except for source and sink. For the correctness proof of the algorithm we refer to Goldberg & Tarjan [1988]. For BRP we slightly change the algorithm by initializing the height of the source to $2m+1$ instead of $|V|$, to improve on time complexity. The algorithm still works, as this height equals the length of the longest path between s and t in the BRP max-flow graph.

To use the parametric max-flow algorithm for BRP the graph should be of a certain form. We discuss the details of the form in the next section when explaining the parametric algorithm. There we show that we should reverse the max-flow graph of BRP to make it obey the constraints needed to use the parametric algorithm. As the complexity results of this section are going to be used in the next, we reverse the graph here. This means that the source and the sink are reversed and the arcs run from source to the disks, to the blocks, and then to the sink.

We continue with the time complexity analysis of the preflow-push algorithm for the reversed graph of BRP for a fixed value of K, following the proof of Goldberg & Tarjan [1988]. Again, we assume that we have a constant number of copies of each block. The initialization can be done in $O(m+n) = O(n)$ time. To bound the complexity of the body of the algorithm we first give a bound on the number of times that the procedures push and lift are called. As the height of the vertices only increases during the algorithm we can bound the number of lift operations by giving an upper bound on the height of a vertex. We first state a lemma that we need in the complexity proof. For the proof of the lemma we refer to Goldberg & Tarjan [1988].

Lemma 3.2. *If f is a preflow and u is a vertex that is overflowing, a path exists from u to s in the residual graph G_f.* □

For the BRP max-flow graph, the maximum length of a shortest path back to the sink is two, due to the bipartite structure of the graph.

Theorem 3.5. *At any time during the execution of the algorithm and for any vertex $u \in V$, $h(u) \leq 2m+3$.*

Proof. First we note that the height of source and sink is constant. For the other vertices the heights are only increased if the vertices are overflowing. So we take any vertex $u \in V \setminus \{s,t\}$ that is overflowing. According to the previous lemma a path of length at most 2 exists from u to s in the residual graph. By the definition of the height labels we know that if (u,v) is a residual arc, then $h(u) \leq h(v)+1$. This gives $h(u) \leq h(s)+2 = 2m+3$. □

Now we can bound the number of times that the procedure lift is called.

Corollary 3.1. *In case $|M_j| = O(1)$, procedure lift is called at most $2m+3$ times per vertex and $(2m+3)(m+n) = O(mn)$ times in total.* □

To bound the number of pushes we split up the total number of pushes in non-saturating and saturating pushes and we give a bound on both.

Theorem 3.6. *The number of non-saturating pushes is $O(m)$.*

Proof. All edges with capacity one are always saturated when used in a push. This means that only on the edges leaving the source non-saturating pushes occur. As these edges are saturated in the initialization we only have to consider pushes back to the source. We adapt the algorithm, such that we do not push flow back to the source before all nodes, except for the nodes connected to the source, have $e(u) = 0$. As the excess of these nodes will not exceed the capacity of the corresponding edge, we do at most one non-saturating push per arc to the source, which gives $O(m)$ non-saturating pushes. □

Theorem 3.7. *The number of saturating pushes is $O(mn)$, in case $|M_j| = O(1)$.*

Proof. The edges leaving the source are used at most twice, once in the initialization, and possibly once to push flow back. This gives at most $2m$ saturating pushes. The edges entering the sink are all used at most once, as they are immediately saturated which gives at most n saturating pushes. Regarding the edges between the disk nodes and the block nodes, they all have capacity one, which means that they are always saturated when used. If flow is pushed over an edge (u,v), it holds that $h(u) = h(v)+1$. Before we can use the edge in opposite direction, i.e. push flow over (v,u), the height of v has to be increased with at least two. By theorem 3.5

we know that the height of a vertex is at most $2m+3$, such that each edge between disk nodes and block nodes is used at most $O(m)$ times. As we have $O(n)$ edges between disk nodes and block nodes, we have $O(mn)$ saturating pushes. □

Concluding from these theorems we can give a bound on the number of calls of the procedures push and lift, the so-called basic operations.

Corollary 3.2. *The preflow-push algorithm solves the decision problem of BRP with $O(mn)$ basic operations, in case $|M_j| = O(1)$.* □

Goldberg & Tarjan [1988] give a sequential implementation of the preflow-push algorithm for which they show that the time complexity equals the number of calls of basic operations, which means that the complexity of the procedures does not increase the complexity of the algorithm. For a complete description of the implementation and the corresponding proofs we refer to their article. Here we sketch the main ideas. They introduce a list Q of overflowing vertices and an fixed ordered edge list for each vertex, that contains undirected edges corresponding to the arcs entering or leaving the vertex. In each list an indicator holds track of the current edge. In each iteration of the algorithm we check for an overflowing vertex v if its current edge can be used for a push. If not, the next edge becomes the current edge and if the end of the list is reached the height of v is increased and the current edge is set to the first edge of the list. In this way a push can be applied in constant time. For the procedure lift we have to run through the edge list once, but lift is only called when the last edge of the edge list is reached. So we can bound the complexity of the algorithm by $O(1)$ per push plus the number of times we run through the edge lists. From Theorem 3.7 we get $O(mn)$ pushes. Running through the edge list of node v takes δ_v time, where δ_v is the degree of v. We know by Theorem 3.5 that the height of each vertex is bounded by $2m+3$. This gives $\delta_v(2m+3)$ per vertex. Summing over the vertices gives $\sum_{v\in V}\delta_v(2m+3) = (2m+3)\sum_{v\in V}\delta_v = (2m+3)2n = O(mn)$. This gives the following theorem.

Theorem 3.8. *The preflow-push algorithm solves the decision variant of the block-based retrieval problem in $O(mn)$ time, in case $|M_j| = O(1)$.* □

For solving the optimization variant, we can do a binary search and add a $\log n$ to the time complexity. However, we show in the next section that it is possible to solve a sequence of max-flow problems with the preflow-push algorithm in the same time complexity as a single problem, if the max-flow graph meets some constraints on the arc capacities. The max-flow graph of BRP satisfies these constraints.

3.2.3 Parametric maximum flow algorithm

In this section we use the work of Gallo, Grigoriadis & Tarjan [1989] on parametric maximum flows to show that the optimization variant of BRP can be solved in $O(mn)$ time. We call a problem a parametric maximum flow problem, if some arc capacities in the max-flow graph depend on a parameter λ. The question is to find a maximum flow that satisfies a second criterion, which is expressed by the λ in the max-flow graph. Solving such a parametric max-flow problem often requires solving a max-flow problem for a sequence of values of λ. Gallo, Grigoriadis & Tarjan [1989] show that this sequence of max-flow problems is solvable in the same time complexity as the max-flow problem for one value of λ if the following constraints hold. The sequence of values of λ is increasing, the capacities of the arcs leaving the source are non-decreasing in λ, those of arcs entering the sink are non-increasing in λ, and all other capacities are constant. The algorithm works as follows.

For the first value of the parameter, λ_1, we compute the maximum flow f_1 with the preflow-push algorithm. Then, we compute the arc capacities for λ_2, where $\lambda_2 > \lambda_1$. The value of λ_2 can be given beforehand or be computed using f_1. We construct a new initial preflow for λ_2 out of f_1 as follows. We set $f_2(u,t)$ to $\min\{c_{\lambda_2}(s,u), f_1(u,t)\}$ for all $(u,t) \in E$ and $f_2(s,u)$ to $\max\{c_{\lambda_2}(s,u), f_1(s,u)\}$ for each arc $(s,u) \in E$ for which $h(u) < h(s)$. The heights of the vertices at the end of the first max-flow computation result in a valid height function for this new preflow, so we leave the heights unchanged and again apply the procedures push and lift until no nodes overflow. This process is repeated until a maximum flow is found for the right value of λ.

In the max-flow graph for BRP, as shown in Figure 3.2, we have the parameter K on the edges towards the sink, so we have a parametric graph. We can easily transform the graph such that it meets the constraints of Gallo et al. We just switch the source and sink and reverse all arcs as done in the previous section. Doing so, we have parametric capacities on the arcs leaving the source and these are non-decreasing in the parameter K. All other arcs have constant capacity. So, the parametric max-flow algorithm can be applied to the max-flow graph of BRP. Now, we show that the parametric algorithm solves BRP in the same time complexity as the preflow-push algorithm for a fixed K.

For BRP, going to the next max-flow problem means that the capacity and the flow of the arcs leaving the source are altered, as these are the only arcs with a parametric capacity, i.e. K. Note that only the arcs that were saturated in the previous step satisfy the constraint $h(u) < h(s)$. We determine the next value of

K in the same way as in the proof of Theorem 3.4. We add to K the number of not-assigned blocks divided by the number of saturated disks. This gives at most m different values of K and the sequence is indeed increasing. The height labels are not influenced and are only increased during the algorithm, such that Theorem 3.5, which gives an upper bound on the height of the vertices, remains valid for the complete parametric algorithm.

We can bound the number of basic operations in the following way. Because Theorem 3.5 is still valid, the number of lift operations remains $O(mn)$. The total number of pushes on arcs leaving the source is bounded by m^2. This can be seen as follows. For each new value of K, a push can occur on these arcs. This is at most m times. The arcs are used backwards at most once. In total this gives $O(m^2)$ pushes. The total number of pushes on arcs to the sink is n, as each arc is used exactly once. For the arcs between the disk nodes and the block nodes we can use Theorem 3.5 in the same way as in the proof of Theorem 3.7. This gives $O(mn)$ pushes. Concluding, the total number of basic operations in the parametric case is still $O(mn)$.

In the same way as before, we can use the sequential implementation with a list Q of overflowing vertices to show that we can actually solve the algorithm in the same time complexity as the number of basic operations. We conclude with the resulting theorem.

Theorem 3.9. *The parametric preflow-push algorithm solves BRP in $O(mn)$ time, in case $|M_j| = O(1)$.* □

3.3 A special case: Random chained declustering

In this last section we consider a special case of duplicate storage, called random chained declustering. It is based on chained declustering as proposed by Hsiao & DeWitt [1990]. They store the successive data blocks in a round-robin fashion, and store for each block that is stored on disk i a copy on disk $(i+1) \bmod m$. Compared to this strategy, we drop the round-robin assignment. We still store two copies of each data block on a pair of disks i and $(i+1) \bmod m$, but choose disk i randomly. For this specific duplicate storage strategy the edge weight partition graph, as introduced in Section 3.1 becomes a cycle, as all edges between non-neighboring disks get a weight zero.

Due to the simple structure of this graph we can design the following linear algorithm to solve the decision variant of BRP. Note that in case of random chained declustering the optimal value of $l_{\max}$ is bounded from below by $\lceil \frac{n}{m} \rceil$ and from

above by $\max w_{ij}$, where the latter results from a clockwise assignment. We use a clockwise point of view and define for each disk i disk $(i+1) \bmod m$ as its successor. For ease of notation we assume in the rest of this section that the operations on the disk numbers are modulo m.

Again we first explain the algorithm that solves the decision variant of BRP for random chained declustering. Theorem 3.10 proves that this algorithm actually solves this problem. Then, we analyze the complexity of the algorithm and the complexity of finding the minimum value of K with a bisection search.

The algorithm starts with an edge with highest weight. Without loss of generality we assume that this edge connects disk 0 and disk 1. We assign K blocks to disk 0 and $w_{0,1} - K \geq 0$ blocks to disk 1. We continue in clockwise direction by defining the following relation for $j = 1, \ldots, m-1$:

$$a_{j+1,j} = \min\{w_{j,j+1}, K - a_{j-1,j}\}, \tag{3.4}$$

$$a_{j,j+1} = w_{j,j+1} - a_{j+1,j}. \tag{3.5}$$

If $a_{j,j+1} > K$ for any disk j, the proof of Theorem 3.10 shows that no feasible solution for this value of K exists. Otherwise, the algorithm finishes the first loop with the computation of $a_{m-1,0}$. At that point there are two possibilities: (*i*) a feasible assignment is constructed, i.e. $a_{j-1,j} + a_{j+1,j} \leq K$ for all $j \in V$, or (*ii*) an overload occurs on disk 0, i.e. $a_{m-1,0} > 0$. In case (*ii*) the algorithm starts a second loop with a new assignment on the first edge. Instead of assigning K blocks to disk 0, we assign $K - a_{m-1,0}$ blocks to disk 0. We recompute the values for each edge with (3.4) and (3.5). Again we conclude infeasibility if $a_{j,j+1} > K$ for any disk j. If the second loop has been completed, we have found a feasible assignment, which is also shown in the proof of Theorem 3.10.

Theorem 3.10. *The double loop algorithm solves the decision variant of BRP for random chained declustering.*

Proof. If the algorithm returns a 'yes' answer, an assignment is given as well, which is correct by construction. In case the algorithm aborts with a 'no' answer, we show that no assignment can be constructed with $l_{\max} \leq K$. We do this by showing that, in that case, a set $V^* \subseteq V$ can be constructed for which $\frac{1}{|V^*|} \sum_{i,j \in V^*} w_{ij} > K$, which is sufficient according to Theorem 3.1.

The algorithm stops if an $a_{j,j+1}$-value is computed that is larger than K. To construct the set V^* we initialize $V^* = \{j+1\}$, move backwards, and add each previous disk to the set V^* if its assigned load equals K, until we reach the first disk with load less than K, say disk i. Note that $i \neq j$. From the construction we know that $a_{i,i+1} = 0$ and that no load can be transferred outside of V^*. For the load within V^*

it holds that

$$\sum_{i_1,i_2 \in V^*} w_{i_1 i_2} = a_{j,j+1} + \sum_{i \in V^* \setminus \{j+1\}} l(i) = a_{j,j+1} + (|V^*| - 1)K > |V^*| \cdot K. \tag{3.6}$$

This implies, according to Theorem 3.1, that no feasible solution exists with $l_{\max} \leq K$.

To complete the proof we show that in case of completion of the second loop always a feasible assignment is found. If the second loop is started, we know that the first loop ended with an overload q on disk 0. For the second loop we start with $a_{1,0} = K - q$ and, consequently, $a_{0,1}$ is increased by q blocks. As $K \geq \lceil \frac{n}{m} \rceil$ and disk 0 had a load larger than K after the first loop, there is at least one disk with a load less than K. We define the set $V_{\min}$ as the set of disks with load less than K and we want to shift the overload on disk 0 to the disks of $V_{\min}$. As $K \geq \lceil \frac{n}{m} \rceil$, we know that $\sum_{i \in V_{\min}} (K - l(i)) \geq q$. During the second loop there are two possible outcomes: (*i*) an $a_{j,j+1}$-value becomes larger than K which means infeasibility or (*ii*) all disks are filled up to K until all q blocks are shifted away to the disks of $V_{\min}$. In the second situation the increase of $a_{0,1}$ does not influence the value of $a_{m-1,0}$, so the latter is still equal to q. The new assignment is feasible as $a_{1,0} = K - q$. □

Theorem 3.11. *The time complexity of the double loop algorithm is $O(m)$.*

Proof. The graph is a cycle of m disks. The algorithm stops after at most 2 loops of m steps each and in each step a constant number of operations has to be executed, which gives the stated result. □

The algorithm for the decision variant can be used to construct a fast algorithm for the optimization variant, by doing a bisection search on the value of K. We know that a feasible K exists in the set $\{\lceil \frac{n}{m} \rceil, \ldots, \max w_{ij}\}$. As the cardinality of this set can be bounded by n, the overall time complexity of the optimization algorithm is $O(m \log n)$.

We saw in the proof of Theorem 3.10 that in case of a 'no' answer a set $V^* \subset V$ can be constructed for which $\frac{1}{|V^*|} \sum_{i,j \in I} w_{ij} > K$. This means that the bisection procedure can be improved by using $\frac{1}{|V^*|} \sum_{i,j \in I} w_{ij}$ as a new lower bound in case of a 'no' answer. This new lower bound makes sure that each time the decision algorithm is run, the algorithm stops at least one node further than in the previous run. This means that we can also bound the number of decision problems to be solved by m, such that the complexity of the optimization algorithm is $O(\min\{m^2, m \log n\})$.

The simplicity of the instance graph in case of random chained declustering enables a very fast algorithm but the freedom for load balancing turns out to be somewhat

smaller, as shown by the simulation results that can be found in [Aerts, Korst & Egner, 2000].

3.4 Discussion

In this section we discuss the difference in input size between BRP and the edge weight partition problem and the consequences on the complexity of the algorithms. Furthermore, we show the correspondence between the proofs of some of the complexity results and the unavoidable load theorem.

Size of the input. The input of BRP is given by a set of blocks and for each block the disks on which it is stored. This means that the size of the input is at least $O(n)$, which is obtained if the size of the sets M_j is bounded by a constant. Given that $m < n$ all presented time bounds are polynomial in the size of the input. For the edge weight partition problem we are given a graph with m nodes and weights on the edges. This means that the size of the input is $O(m^2 \log n)$ for random duplicate storage and $O(m \log n)$ for random chained declustering. Consequently, the complexity bounds of the algorithms that all contain a factor n are not polynomial for the edge weight partition problem. This is a note of mainly theoretical importance, as from the application a linear correspondence between n and m can be derived.

Unavoidable load. The instance graph and the corresponding unavoidable load theorem are strong instruments for the analysis of BRP. In several proofs in this chapter we used the unavoidable load argument. In the proof of theorem 3.4, which gives the complexity of the Dinic-Karzanov algorithm, we describe in the second part a way of updating K such that at most m updates are necessary. The nodes corresponding to the edges towards the sink that are not saturated do not belong to the subset that determines the optimum value of K. The saturated edges in the last step correspond to the subset that determines the value of the optimal load. In the proof of Theorem 3.10, which proves the correctness of the double loop algorithm, we explicitly used the unavoidable load of a subset.

4

Time-Based Load Balancing

In the time-based retrieval problem we take the actual transfer times and the switch times into account when minimizing the period length, which is defined as the completion time of the last disk. Again a number of blocks has to be retrieved from a number of disks. For each block the subset of disks is given on which the block is stored. Furthermore, for each block the transfer time is given for each disk on which the block is stored. The problem is to assign (parts of) blocks to the disks such that the period length is minimized. Compared to BRP, where we minimize the number of blocks assigned to one disk, the advantage of the time-based approach is that we can exploit the multi-zone character of disks and the possibility to read a block in parts from several disks. This gives better load balancing results and more efficient usage of the disks. Furthermore, in this model heterogeneous streams and heterogeneous disks can be embedded, which makes time-based load balancing applicable to a broader range of system settings than block-based load balancing.

This chapter is organized as follows. In Section 4.1 we introduce a mixed integer linear programming formulation for TRP. We analyze the computational complexity of TRP in Section 4.2. We prove that TRP is NP-complete in the strong sense and analyze the complexity of some special cases. In Section 4.3 we introduce

several algorithms for TRP. We also give performance bounds for these algorithms. The first three sections deal with RMS for homogeneous streams as well as disks. In Section 4.4 we discuss the application to other storage strategies. The chapter ends with a discussion section where we discuss the applicability to heterogeneous settings.

4.1 TRP modeling: An MILP formulation

To minimize the completion times of the disks, we take the actual transfer times of the blocks into account and embed the switch time into the model. Furthermore, we introduce the possibility of partial retrieval. First, we restate the problem formulation that was given in Section 2.4, and afterwards we model TRP as an MILP problem.

We are given a set J of n data blocks to be retrieved from a set M of m disks and for each block j a set M_j of disks on which block j is stored. For each disk i and block j, we introduce a parameter u_{ij} which is one if $i \in M_j$ and zero otherwise. The transfer time to retrieve block j from disk i is given by p_{ij}. Furthermore, the total switch time of disk i is approximated by $n_i s + c$, where n_i is the number of blocks assigned to disk i. The switch slope s and the switch offset c are given.

We introduce for all $j \in J$ and $i \in M$ a decision variable x_{ij}, indicating the fraction of block j to be retrieved from disk i. Associated with each x_{ij} is a binary variable $y_{ij} = \lceil x_{ij} \rceil$, indicating whether or not block j is (partially) retrieved from disk i. We denote the period length by $T_{\max}$. Then, we can formulate the time-based retrieval problem as the following MILP problem.

$$\begin{aligned}
\min \quad & T_{\max} && && (4.1)\\
\text{s.t.} \quad & \sum_{j \in J} x_{ij} p_{ij} + s \sum_{j \in J} y_{ij} + c \leq T_{\max} && \forall i \in M && (4.2)\\
& \sum_{i \in M} x_{ij} = 1 && \forall j \in J && (4.3)\\
& 0 \leq x_{ij} \leq u_{ij} && \forall j \in J,\, i \in M && (4.4)\\
& y_{ij} \geq x_{ij} \ \wedge\ y_{ij} \in \{0,1\} && \forall j \in J,\, i \in M && (4.5)
\end{aligned}$$

4.2 Complexity of TRP

In this section we analyze the computational complexity of the time-based retrieval problem. The results in this section are based on the work done by Aerts, Korst, Spieksma, Verhaegh & Woeginger [2002]. We start with proving that the decision

variant of TRP is NP-complete in the strong sense by a reduction from 3-partition, which is known to be NP-complete in the strong sense [Garey & Johnson, 1979]. To represent the retrieval problems we use the multiprocessor scheduling notation as introduced in Section 2.4.

Problem 5 [3-Partition]. Given are a set of items $A = \{1, \ldots, 3k\}$ with sizes $a_1, \ldots, a_{3k}$ and a bound B, for which $\frac{B}{4} < a_i < \frac{B}{2}$ for all i and $\sum_i a_i = kB$. The question is whether or not A can be partitioned into k subsets, such that the sum of the item sizes of each subset equals B. □

Theorem 4.1. *The decision variant of TRP, i.e.* $R\,|\,M_j, pmtn^*, set\text{-}up\,|\,C_{\max} \leq T$*, is NP-complete in the strong sense.*

Proof. It is obvious that we can check in polynomial time for a given assignment whether or not all disks are finished at time T, so the problem is an element of the class NP. To show that the problem is NP-complete in the strong sense we show that a polynomial time reduction exists from 3-partition to TRP. We note that in this reduction the largest number of the TRP instance is polynomially bounded by the largest number of the 3-partition instance.

Considering an instance of 3-partition, we construct an instance of TRP in the following way. We take k disks and define for each number a_j of the 3-partition instance a block j, which is stored on all disks, i.e. $u_{ij} = 1$ for all $i \in M$, and has a transfer time $p_{ij} = p_j = a_j$ on each disk i. Furthermore, we define the time bound of TRP as $T = 4B$, and the values s and c of the switch time function as B and 0, respectively. Now we show that a positive answer for 3-partition is equivalent to a positive answer for TRP.

$\Rightarrow$ Given a solution to the 3-partition instance, we assign each subset to a different disk. For each disk the sum of the transfer times equals B and three y-values equal one, so the completion time for each disk equals $B + 3s = 4B$.

$\Leftarrow$ Assume we have an assignment for TRP with value $4B$. As the transfer times are strictly larger than zero and $s = B$, no disk retrieves more than three blocks, which means that no blocks are preempted. Consequently each disk retrieves exactly three blocks. Combining this with the facts that $\sum p_i = kB$ and no disk exceeds $4B$ we conclude that the blocks assigned to each disk form a feasible subset in 3-partition. □

Note that the above construction also proves that $P\,|\,M_j\,|\,C_{\max} \leq T$ is NP-complete in the strong sense. In the theorem we did not put any restriction on the sets M_j and we used $M_j = M$ for all $j \in J$. Concluding from that we can also state the following corollary for a special case of TRP.

Corollary 4.1. *The decision variant of TRP is NP-complete in the strong sense, even if all blocks are stored on all disks and each block has the same transfer time on all disks, i.e.* $p_{ij} = p_j$ *for all i.* □

From the above results we cannot conclude that TRP for RDS is NP-complete in the strong sense, as it might be the case that this restriction on the sets M_j makes the problem easier. However, the next theorem proves with a reduction from a specific variant of the satisfiability problem that this is not the case. We first introduce this variant of satisfiability, which is NP-complete in the strong sense as proved by Tovey [1984].

Problem 6 [Tovey-SAT]. Given a collection of clauses on a finite set of boolean variables, where each clause consists of two or three variables and each variable occurs at most three times, can we find a truth assignment to the variables that satisfies all the clauses? □

We note that in an instance of Tovey-SAT each variable occurs at most twice in the positive form and at most twice in the negative form without loss of generality. This can be seen as follows. In case a variable occurs only in one form, positive or negative, the corresponding literals and clauses can be omitted from the instance by choosing a trivial truth assignment for the variable, that makes the literals and the clauses true.

Theorem 4.2. *The decision variant of TRP for RDS, or equivalently* $R\,|\,|M_j| = 2$, *pmtn**, *set-up*$\,|\,C_{\max} \leq T$, *is NP-complete in the strong sense.*
Proof. We prove that $R\,|\,|M_j| = 2$, pmtn*, set-up$\,|\,C_{\max} \leq 2$ is NP-complete in the strong sense, which implies the theorem.

We first prove that a polynomial time reduction from Tovey-SAT to $P\,|\,|M_j| = 2\,|\,C_{\max} \leq 2$ exists. Next, we show that the problem with preemption and set-up times is at least as difficult, which implies that the decision variant of TRP for RDS is NP-complete in the strong sense.

We translate an instance of Tovey-SAT into an instance of $P\,|\,|M_j| = 2\,|\,C_{\max} \leq 2$ in the following way. An instance of Tovey-SAT consists of variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$. We define the TRP instance as follows.

- For each variable x_i we define two disks, $m(x_i)$ and $m(\overline{x_i})$, and a block $j(x_i)$ with transfer time two which is stored on $m(x_i)$ and $m(\overline{x_i})$.
- For each clause c_j with three elements we define a disk $m(c_j)$.
- For each clause c_j with two elements we define three disks, $m(c_j)$, $m_1(c_j)$, and $m_2(c_j)$, and three blocks. The first two blocks, $j_1(c_j)$ and $j_2(c_j)$, have

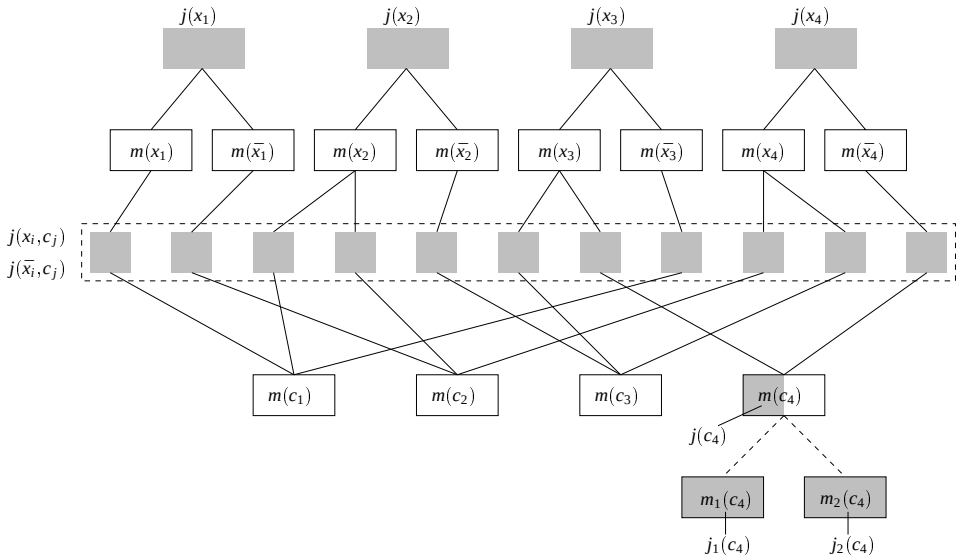


Figure 4.1. Instance of TRP for RDS corresponding to Tovey-SAT instance given by $(x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee x_2 \vee x_4) \wedge (\bar{x_2} \vee x_3 \vee x_4) \wedge (x_3 \vee \bar{x_4})$.

transfer time two and are stored on disks $m_1(c_j)$ and $m_2(c_j)$. The third block $j(c_j)$ has transfer time one and is stored on $m(c_j)$ and $m_1(c_j)$.

- For each positive occurrence of variable x_i in clause c_j we define a block $j(x_i, c_j)$ that has transfer time one and is stored on disks $m(c_j)$ and $m(x_i)$. For each negative occurrence of x_i, i.e. $\bar{x_i}$ a block $j(\bar{x_i}, c_j)$ is constructed with transfer time one that is stored on disks $m(c_j)$ and $m(\bar{x_i})$.

Figure 4.1 shows an example of the translation. Note that due to the construction the disks $m_1(c_j)$ and $m_2(c_j)$ are completely filled by $j_1(c_j)$ and $j_2(c_j)$. Consequently, a disk $m(c_j)$ corresponding to a clause with two elements, like c_4 in the figure, has only space for one block with transfer time one. In the figure we already assigned these jobs to the machines.

Now we show that a feasible truth assignment for Tovey-SAT is equivalent to a feasible solution for $P\big||M_j| = 2\big|C_{\max} \leq 2$.

$\Rightarrow$ Assume that we have a feasible truth assignment. Considering a variable x_i with the value 'true', we (*i*) retrieve block $j(x_i)$ from disk $m(\bar{x_i})$, (*ii*) retrieve the blocks $j(x_i, c_j)$ corresponding to positive occurrences of x_i from disk $m(x_i)$ (note that at most two of these exist), and (*iii*) retrieve the blocks $j(\bar{x_i}, c_j)$ from disk $m(c_j)$. For a variable with value 'false' the assignments are the other way around, which means that block $j(x_i)$ is retrieved from disk $m(x_i)$, blocks $j(x_i, c_j)$ corresponding

to positive occurrences of x_i from disk $m(c_j)$, and blocks $j(\overline{x_i}, c_j)$ from disk $m(\overline{x_i})$.

In short, it means that the assignment of the blocks $j(x_i)$ is opposite to the thruth assignment and that the blocks $j(x_i, c_j)$ and $j(\overline{x_i}, c_j)$ that are retrieved from disks $m(x_i)$ or $m(\overline{x_i})$ make the corresponding clauses true. The disks $m(x_i)$ and $m(\overline{x_i})$ get assigned at most two units of transfer time as can be seen from (*i*) and (*ii*), and, as each clause c_j is satisfied, this also holds for the disks $m(c_j)$. So the above assignment gives a feasible schedule for $P\big||M_j| = 2\big|C_{\max} \leq 2$.

$\Leftarrow$ Assume that we have a feasible schedule for TRP. We assign a variable x_i the value 'true' in case block $j(x_i)$ is retrieved from disk $m(\overline{x_i})$, and 'false' otherwise. This means that the variables corresponding to the blocks $j(x_i, c_j)$ and $j(\overline{x_i}, c_j)$ that are scheduled on the disks $m(x_i)$ and $m(\overline{x_i})$, make their clauses true. As in the schedule no overload occurs on the clause disks $m(c_j)$, for each clause at least one of the blocks $j(x_i, c_j)$ or $j(\overline{x_i}, c_j)$ is scheduled on $m(x_i)$ or $m(\overline{x_i})$, such that we have constructed a feasible truth assignment for Tovey-SAT.

From the above we conclude that the non-preemptive case is NP-complete in the strong sense. It is now sufficient to show that the problem with preemption and set-up times is at least as difficult. We do this by showing that the correspondence explained above still holds in case of preemption and a set-up time of $\frac{3}{4}$. Furthermore, we change the transfer times of blocks $j(x_i)$ with transfer time two into a new transfer time of $\frac{5}{4}$ and of the other blocks into a transfer time of $\frac{1}{4}$.

Due to the set-up times, the only way to retrieve the blocks $j_1(c_j)$ and $j_2(c_j)$ before time two is as depicted in Figure 4.1, which means that these blocks cannot be preempted in a feasible schedule.

We now show that without loss of generality we may assume that the blocks $j(x_i)$ are not preempted. Assume that one of the large blocks $j(x_i)$ is preempted. This means that it takes a set-up time of $\frac{3}{4}$ on both $m(x_i)$ and $m(\overline{x_i})$. The transfer time of the block is $\frac{5}{4}$ and has to be divided over both disks. At least on one of the disks the remaining idle time is less than $\frac{3}{4}$, such that no second block can be retrieved from that disk, again due to the set-up time. Consequently, it is better to retrieve block $j(x_i)$ entirely from that disk and leave the other disk empty.

As the larger blocks $j(x_i)$ are not preempted, all blocks with transfer time $\frac{1}{4}$ are retrieved from disks from which only similar small blocks are retrieved. It is always possible to retrieve two of these small blocks from one disk, but it is impossible to retrieve (parts of) three blocks from one disk, as three times the set-up time is larger than 2. This means that these small blocks are neither preempted.

We conclude that preemption with set-up times does not change the complexity of

the problem, such that $P\big||M_j| = 2$, pmtn*, set-up$\big|C_{\max} \leq 2$ is NP-complete in the strong sense. The case of independent disks, i.e. $R\big||M_j| = 2$, pmtn*, set-up$\big|C_{\max} \leq 2$ is a generalization. □

It is a well-known result that multiprocessor scheduling problems with preemption but without set-up times can be modeled as a linear programming problem and consequently are solvable in polynomial time. Machine eligibility constraints fit in such an LP model as can be seen from the MILP formulation of TRP. For the sake of completeness we hence add the following corollary.

Corollary 4.2. *$R\big|M_j$, pmtn* $\big|C_{\max}$ is solvable in polynomial time.* □

In general, complexity results for multiprocessor scheduling problems change if the number of processors is considered to be a part of the problem definition instead of part of the input. In the remainder of this section we focus on retrieval problems with a fixed number of disks. These problems are of practical interest, as they describe the retrieval problems for a given disk array. We start with a complexity analysis of the problems without preemption. We first prove that $P\big||M_j| = 2$, set-up$\big|C_{\max} \leq T$ is NP-complete by a reduction from partition, which is NP-complete in the ordinary sense [Garey & Johnson, 1979].

Problem 7 [Partition]. Given are a set of items $A = \{1, \ldots, k\}$ with sizes $a_1, \ldots, a_k$ and a bound $B = \frac{1}{2}\sum_i a_i$. The question is whether or not A can be partitioned into two subsets such that the sum of the elements of each subset equals B. □

Theorem 4.3. *$P2\big||M_j| = 2$, set-up$\big|C_{\max} \leq T$ is NP-complete.*

Proof. We define the correspondence between partition and the scheduling problem as follows. For each number a_j of the partition problem we define a block j with $p_j = a_j$, which can be retrieved from both disks. The time bound T of the scheduling problem equals B. It is straightforward to see that both problems are equivalent. □

Note that in this case neither the set-up times nor the eligibility constraints $|M_j| = 2$ add anything to the problem, which makes the above proof also applicable for $P2\big|\big|C_{\max} \leq T$.

The proof of Theorem 4.3 applies to a number of retrieval problems. To prove that $Pm\big||M_j| = 2$, set-up$\big|C_{\max} \leq T$ is NP-complete, we can use the same k blocks as in the proof of Theorem 4.3. The other $m-2$ machines are left idle. Further generalization gives that $Pm\big|M_j$, set-up$\big|C_{\max} \leq T$ and $Rm\big|M_j$, set-up$\big|C_{\max} \leq T$ are NP-complete as well.

The same proof holds for the above problems with preemption by taking a positive set-up time $s < \min a_j$ and transforming the transfer times into $p_j = a_j - s$. Observe that the sum of the transfer times equals $2B$, such that the first two machines are completely filled without preemptions, and consequently no blocks can be preempted, due to the positive set-up time.

So far, all problems with a fixed number of disks are shown to be NP-complete in the ordinary sense. This means that the best we can get regarding an optimization algorithm is an algorithm that runs in pseudo-polynomial time, which means that the runtime can be bounded by a polynomial in the size of the input and the largest number in the input. We describe such a pseudo-polynomial time algorithm for this set of retrieval problems. We start with the assumption that all transfer times and the parameters s and c of the switch time function are integer. Then, the following theorem holds.

Theorem 4.4. *$Rm\,|\,M_j$, set-up $|\,C_{\max} \leq T$ is solvable in pseudo-polynomial time.*

Proof. We prove this theorem by giving an algorithm with a time complexity that is bounded by a polynomial in the size of the input and the largest number in the input. The algorithm is a generalization of a dynamic programming algorithm for the knapsack problem [Martello & Toth, 1990]. We assign the blocks one by one according to a given block list.

We try to solve the question whether or not we can find a schedule that is finished at time T. We represent the state of the algorithm by a vector $(x_1, \ldots, x_m)$, where $x_i \in \mathbb{N}$ denotes the amount of transfer time plus switch time assigned to disk i. We can restrict ourselves to states for which $0 \leq x_i \leq T$ for all i, such that the number of possible states equals $(T+1)^m$.

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

$$F_k = \left\{ x + p_{ik} e_i \,\middle|\, x \in F_{k-1} \wedge i \in M_k \right\}, \tag{4.6}$$

where e_i is the i^{th} unit vector. We omit the states in which any of the values x_i is larger than T, as these states never lead to a feasible assignment.

Now the decision problem can be reformulated as follows. A feasible assignment exists if and only if $F_n \neq \emptyset$. The complexity of this algorithm is bounded by $O(T^m \cdot n \cdot m)$, with m being a constant, so it is polynomial in the size of the input and T. □

A similar dynamic programming algorithm can be constructed for the following problems, because these are all special cases of $Rm\,|\,M_j$, set-up$\,|\,C_{\max} \leq T$.

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

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

We end this section by proving that $Rm\,|\,M_j$, pmtn*, set-up$\,|\,C_{\max}$ is solvable in pseudo-polynomial time as well. Recall from the MILP formulation that a schedule is fully specified by non-negative values x_{ij} for all $i \in M$, $j \in J$, for which $\sum_{i=1}^{m} x_{ij} = 1$. A set-up time s is added to disk i if and only if $x_{ij} > 0$, i.e. $y_{ij} = 1$. A block j is preempted if the value of some of its variables x_{ij} lie strictly between 0 and 1. We first prove a lemma that bounds the number of preemptions in an optimal schedule in which the number of preemptions is minimized.

Lemma 4.1. *For any instance of $Rm\,|\,M_j$, pmtn*, set-up$\,|\,C_{\max}$, an optimal schedule exists with at most $\frac{1}{2}m(m-1)$ preempted blocks.*

Proof. Among all optimal schedules, select an optimal schedule σ with the smallest number of preempted parts, i.e. variables x_{ij} with a positive value. Each preempted block in σ occupies at least two disks. If more than $\frac{1}{2}m(m-1)$ blocks are preempted, at least two of them occupies at least two common disks, as at most $\binom{m}{2} = \frac{1}{2}m(m-1)$ different pairs of disks exist. We show that this contradicts the assumption that σ has the smallest number of preempted parts, which proves the lemma.

Consider in σ the blocks j and j' and the disks i and i' for which the values x_{ij}, $x_{i'j}$, $x_{ij'}$, $x_{i'j'}$ are all non-zero. Without loss of generality we assume that $p_{ij}/p_{ij'} \geq p_{i'j}/p_{i'j'}$. We next show that we can reduce at least one of the x-values to zero, without losing optimality of the schedule. We do this by shifting an amount $\varepsilon = \min\{x_{ij}, x_{i'j'}\, p_{ij'}/p_{ij}\}$ of block j from disk i to i' and an amount $\delta = \varepsilon p_{ij}/p_{ij'}$ of block j' back from disk i' to i. More formally, we construct a schedule σ' that results from σ by setting

$$x'_{ij} = x_{ij} - \varepsilon, \qquad x'_{ij'} = x_{ij'} + \delta,$$
$$x'_{i'j} = x_{i'j} + \varepsilon, \qquad x'_{i'j'} = x_{i'j'} - \delta.$$

Note that σ' is still feasible as all values are nonnegative. The total load on disk i changes by

$$-\varepsilon p_{ij} + \delta p_{ij'} \;=\; -\varepsilon p_{ij} + \varepsilon p_{ij} \;=\; 0.$$

The total load on disk i' increases by

$$\varepsilon p_{i'j} - \delta p_{i'j'} \;=\; \varepsilon p_{i'j} - \varepsilon p_{i'j'} p_{ij} / p_{ij'} \;\leq\; \varepsilon p_{i'j} - \varepsilon p_{i'j'} p_{i'j} / p_{i'j'} \;=\; 0.$$

Hence, the makespan of schedule σ' is at most the makespan of schedule σ, and as σ is an optimal schedule, σ' is optimal as well. By the definition of ε and δ, at least one of the values x'_{ij} and $x'_{i'j'}$ equals zero. Thus, σ' is an optimal schedule with a smaller number of preempted parts, which contradicts the definition of σ. □

Theorem 4.5. *$Rm \,|\, M_j, pmtn^*, set\text{-}up \,|\, C_{\max}$ can be solved in pseudo-polynomial time.*

Proof. By Lemma 4.1 it is sufficient to search within the set S of schedules with at most $\frac{1}{2}m(m-1)$ preempted blocks. We partition S into classes of similar schedules; two schedules belong to the same class if and only if they preempt exactly the same blocks and assign the parts to exactly the same disks. There are only $O(n^{m(m-1)/2})$ possibilities for selecting the preempted blocks from a total of n blocks. For each preempted block, there are at most 2^m possibilities for assigning it to a subset of the disks. Hence, the overall number of such classes is bounded from above by $O(n^{m(m-1)/2} 2^{m^2(m-1)/2})$, which is a polynomial in n, as m is constant.

We now show how to compute an optimal schedule from a fixed class. We start with generating all possible load vectors for the non-preempted blocks in this class, where a load vector specifies for each disk the total amount of assigned transfer time including set-up time. This can be done in pseudo-polynomial time by standard dynamic programming, in a similar way as in the proof of Theorem 4.4. It remains to add the load of the preempted blocks to the load vectors. For each preempted block j, let M_j^* be the set of disks to which it is assigned, according to the class under consideration. Now, block j adds a set-up time to each of the disks $i \in M_j^*$. Furthermore, it adds a transfer time $x_{ij} p_{ij}$ to each disk $i \in M_j^*$. Obviously, we must have $\sum_{i \in M_j^*} x_{ij} = 1$, and $x_{ij} = 0$ for all $i \notin M_j^*$. So, the problem to determine the x_{ij} values of the preempted blocks can be formulated as a linear program, which can be solved in polynomial time.

The final output is the best solution that we find over all classes. This can be done in pseudo-polynomial time, which proves the theorem. □

In Figure 4.2 we give an overview of the complexity results that are derived in this section. For completeness sake we also include the block-based problems

discussed in Chapter 3.

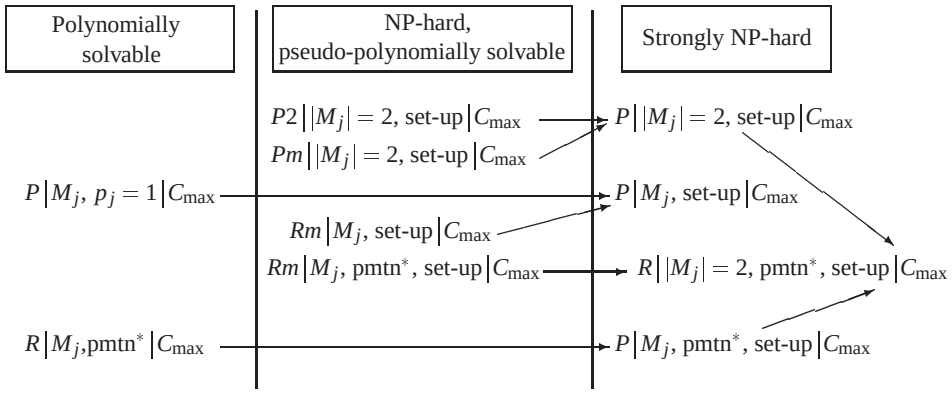


Figure 4.2. Complexity diagram of retrieval problems. The arrows indicate relationships between the problems, in the sense that adding or generalizing a job or machine characteristic transforms the first one into the other one. Arrows that cross a vertical line correspond to a generalization or specification that makes the problem harder.

From the figure, we can conclude that the retrieval problems with unit-processing times or with preemption without set-up times are solvable in polynomial time. The problems with a fixed number of disks are all pseudo-polynomially solvable. Dropping these three assumptions makes the problems NP-hard in the strong sense. This is indicated in the figure with the arrows, which are all directed from one retrieval problem to a more generalized one. Furthermore, we see that the eligibility constraints do not influence the complexity of the problems. The complexity results are in line with results for multiprocessor scheduling problems without eligibility constraints.

4.3 Algorithms for TRP

In this section we present algorithms for TRP. As TRP is proven to be NP-complete we cannot expect to find a polynomial time optimization algorithm. We first present two approximation algorithms that use the solution of an LP-relaxation. Then, we introduce a list scheduling heuristic and a postprocessing procedure, that can be used to improve non-preempted solutions.

4.3.1 LP rounding

A straightforward way to derive an algorithm for TRP is by solving its LP-relaxation and rounding up the y-variables. Without loss of generality we can restrict ourselves to solutions of the LP-relaxation where each y-variable has the same value as the corresponding x-variable, as we want to minimize the period length and $s \geq 0$. This means that we can omit the y-variables from the formulation. We use an LP-solver to solve the resulting LP-problem, which is formulated as follows.

$$\begin{aligned}
&\min && T_{\max} && && (4.7)\\
&\text{s.t.} && \sum_{j \in J} x_{ij}(p_{ij}+s)+c \leq T_{\max} && \forall i \in M && (4.8)\\
& && \sum_{i \in M} x_{ij} = 1 && \forall j \in J && (4.9)\\
& && 0 \leq x_{ij} \leq u_{ij} && \forall j \in J,\, i \in M && (4.10)
\end{aligned}$$

The so-called LP rounding algorithm for TRP works as follows. It solves the LP-relaxation, rounds up the y-variables, and computes the actual cost with

$$T_{\max} = \max_{i \in M} \sum_{j \in J} x_{ij} p_{ij} + s \sum_{j \in J} y_{ij} + c. \tag{4.11}$$

We denote for an instance I the cost of a solution of LP rounding by $S_{\text{round}}(I)$, the cost of an optimal solution by $S_{\text{opt}}(I)$, and the cost of the outcome of the LP-relaxation by $S_{\text{LP}}(I)$. The following theorem gives a performance bound for LP rounding.

Theorem 4.6. *For each instance I of TRP we have*

$$\frac{S_{\text{round}}(I)}{S_{\text{opt}}(I)} \leq 1 + \frac{m \cdot s}{\frac{n}{m} \cdot (p_{\min}+s)+c}, \tag{4.12}$$

where $p_{\min}$ equals the transfer time of a block in the innermost zone.

Proof. First we give an upper bound on the number of preemptions, as non-integral y-variables cause the increase in the actual cost, compared to the cost of a solution of the LP-relaxation. The number of non-zero variables in a solution of a linear programming problem, when using the simplex method, is at most the number of constraints. In the LP-relaxation of the retrieval problem we have the constraints (4.8) and (4.9) which are $m+n$ constraints. As for each $j \in J$ at least one x_{jm} should be larger than 0, this implies that the number of preemptions is at most m. So, $S_{\text{LP}}(I) + m \cdot s$ is an upper bound for the solution value of LP rounding.

Furthermore, note that $S_{\mathrm{LP}}(I)$ is a lower bound on the optimal cost of instance I and

$$S_{\mathrm{LP}}(I) \geq \frac{n}{m}(p_{\min} + s) + c. \tag{4.13}$$

With these bounds we get

$$\frac{S_{\mathrm{round}}(I)}{S_{\mathrm{opt}}(I)} \leq \frac{S_{\mathrm{LP}}(I) + m \cdot s}{S_{\mathrm{LP}}(I)} = 1 + \frac{m \cdot s}{S_{\mathrm{LP}}(I)} \leq 1 + \frac{m \cdot s}{\frac{n}{m}(p_{\min} + s) + c}.$$

□

In practice, the ratio between n and m depends on the ratio between disk transfer rate and consumption rate, which gives an indication for the number of clients that can be served by one disk. For a given set of system parameters this ratio is more or less constant, and consequently, the m factor in the numerator of (4.12) makes that the performance bound grows in the size of the system. In the next section we describe an algorithm that does not have this disadvantage.

4.3.2 LP matching

In this section we derive a second approximation algorithm based on LP-relaxation. We follow the work of Lenstra, Shmoys & Tardos [1990] who use an LP-relaxation to solve the non-preemptive scheduling problem $R\big|\big|C_{\max}$. With a matching description of the preempted jobs of the LP-solution, they prove that a non-preemptive solution can be constructed out of the LP-solution in which each machine gets assigned at most one of the blocks that was preempted in the LP-solution. For TRP this means that the increase in cost on top of the cost of the LP solution is at most $p_{\max} + s$, where $p_{\max}$ denotes the maximum transfer time. This algorithm is called LP matching. We denote a solution of the LP matching algorithm by S_{match} and derive a performance bound as follows.

Theorem 4.7. *For each instance I of TRP we have*

$$\frac{S_{\mathrm{match}}(I)}{S_{\mathrm{opt}}(I)} \leq 1 + \frac{p_{\max} + s}{\frac{n}{m}(p_{\min} + s) + c}. \tag{4.14}$$

Proof. From the proof of Theorem 4.6 and the fact that the matching adds at most $p_{\max} + s$ to the LP-solution the stated result follows immediately. □

4.3.3 List scheduling heuristic

Next to the two above approximation algorithms we introduce a list scheduling heuristic for TRP. This is a time-based version of the linear reselection heuristic as introduced by Korst [1997]. The algorithm is comparable to shortest queue scheduling. It assigns the blocks one by one to the disks according to a given block list. Each block j is assigned to a disk $m \in M_j$ for which the resulting load is minimal, where the resulting load is defined as the currently assigned load plus the load that would result from the assignment of j. In a second round we reconsider all blocks and check for each block j if reassigning it results in a lower value of $\max_{i \in M_j} l(i)$. If so, we reassign the block.

4.3.4 Postprocessing

The LP matching algorithm as well as the list scheduling heuristic result in a solution without preempted blocks. To improve the solution we can perform a postprocessing step where we allow preemption again. We do this by trying to preempt each block $j = 1, \ldots, n$ in such a way that the workload of its disks is more balanced. For duplicate storage we do the following. Consider a block j for which a request is assigned to disk i_1 and which is also stored on disk i_2. We reassign a fraction $x = \min\{1, \frac{l(i_1)-l(i_2)-s}{p_{i_1 j}+p_{i_2 j}}\}$ from disk i_1 to disk i_2 if this fraction $x > 0$. The solution after the postprocessing step is at least as good as the outcome without postprocessing, so for LP matching with postprocessing the performance bound of Theorem 4.7 remains valid.

4.4 Random multiplication and random striping

The load balancing approach that we presented in this chapter is applicable to a broad range of storage strategies and system settings. In this last section of the chapter we show how the models and algorithms for TRP work for other storage strategies.

First of all, we note that the MILP formulation as stated in Section 4.1 is valid for partial duplication and other multiplication strategies, such as triple storage. All these strategies can be modeled by choosing an appropriate value for each u_{ij} and the LP-based approximation algorithms can be used for solving the problem. The postprocessing procedure can be redefined such that it holds for other multiplication storage strategies than duplicate storage.

In case of random striping we have to adapt the MILP formulation to subblocks.

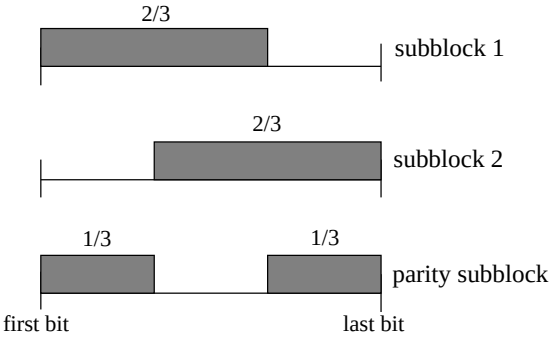


Figure 4.3. Example shows that for random striping with preemption it can be impossible to retrieve the three parts with one disk access per part.

One way to do this is by redefining (4.3) as $\sum_i x_{ij} = r$ for each block j, where r is the parameter of random striping. Then, the other constraints and the optimization criterion remain unchanged if we consider p_{ij} to be the transfer time of a subblock. However, if we still allow fractional values for x_{ij}, it might be impossible to retrieve each fraction with only one access, as can be seen from Figure 4.3.

The figure shows for $r = 2$ a block for which each corresponding x-variable gets assigned a value $2/3$. The values sum up to two, but it is impossible to retrieve each subblock with only one access, in such a way that the original block can be reconstructed out of the parts, as two of the subblocks need the first bit and two of the subblocks need the last. This means that the linear estimation of the switch time is no longer an upper bound, as the number of accesses per disk can no longer be computed with the y-variables. One way to get a feasible ILP model is by omitting preemption, i.e. by adding the integrality constraint $x_{ij} \in \{0,1\}$ for all i and j. This results in the following ILP model.

$$\begin{aligned}
\min \quad & T_{\max} && && (4.15)\\
\text{s.t.} \quad & \sum_{j \in J} x_{ij}(p_{ij}+s) + c \leq T_{\max} && \forall i \in M && (4.16)\\
& \sum_{i \in M} x_{ij} = r && \forall j \in J && (4.17)\\
& 0 \leq x_{ij} \leq u_{ij} && \forall j \in J,\, i \in M && (4.18)\\
& x_{ij} \in \{0,1\} && \forall j \in J,\, i \in M && (4.19)
\end{aligned}$$

In a solution to this ILP model no jobs are preempted. To improve a non-preempted solution we could use the observation that we can reconstruct the original block if at most two of the x-values are fractional. The algorithm that we propose for

random striping works then as follows. We drop constraint (4.19) to get an LP problem and perform a rounding procedure on the fractional LP solution. During rounding we make sure that for each j at most two x_{ij}-values are fractional. In the simulation experiments that are described in the next chapter, we implemented random striping for $r = 2$. The rounding procedure that we implemented works as follows. For each preempted job we check if the number of fractional values is two. If so, we leave the x-values unchanged and add the remaining part of s to the load of the corresponding disk. This is equivalent to rounding the y-values in case of RDS. If all three values are fractional, we take the largest and round the corresponding x-value up to one and subtract this part from one of the other fractions. If one of the others is smaller than this fraction we take that one, otherwise we subtract it from the slowest subblock. Note that rounding a variable x_{ij} increases the load of disk i with at most $\frac{1}{3}(p_{ij} + s)$.

Next to this LP rounding algorithm we implemented a list scheduling heuristic. In this heuristic we initially assign the two fastest subblocks to the corresponding disks. Then, we check if reassigning a subblock results in an improvement. For possible reassignment, we consider the blocks in the following order. We start with the blocks for which the largest transfer time is smallest. For these blocks the difference between this slowest subblock and the other two blocks is smallest. We first check if dropping the second slowest subblock and using the slowest one results in an improvement. Otherwise we check if reassigning the fastest subblock results in an improvement. In this way we check for all blocks if reassigning one of the subblocks results in an improvement. The algorithm performs even better if we do a second run of reassignments.

4.5 Discussion

So far we discussed in this thesis homogeneous settings, where the disks of the disk array are all identical and the streams requested by the clients have the same maximum bit-rate. For the application of BRP, which uses unit transfer times, this homogeneous setting is essential. However, for TRP, where we can take the actual transfer time of each block-disk combination into account, we can drop these assumptions and apply models and algorithms similar to the ones described in this chapter. In this discussion section we give an idea of how the models should be adapted.

Heterogeneous disks. In case of heterogeneous disks the MILP model remains valid and the introduced algorithms can thus be used. We still use constant data length blocks, so a fast disk can retrieve in each period more blocks than a slower

one. The storage strategy becomes a bit more complicated as we should take the disk speed into account. The retrieval algorithms automatically assign the workload according to disk speed, as they try to minimize the period length. Note that in this case the parameters of the switch time function become disk dependent parameters.

Heterogeneous streams. A second generalization is a video-on-demand system that offers streams at different bit-rates, e.g. due to different quality levels. Recall that in the design of a homogeneous system the block size was related to the period length. In case of heterogeneous streams we have to configure the system according to a certain period length and determine a block size for each stream individually. Again, the block size of each stream is large enough to provide video in a worst-case period. The number of streams that can be admitted depends on the bit-rates required for the streams. In a highly loaded system it is possible that a newly requested stream is only admitted if it is a low bit-rate stream. The models and algorithms discussed in this chapter can be used to configure the system, to do the admission control, and to distribute the load in each cycle when the system is running, in the same way as for homogeneous streams.

5

Performance Analysis

In this chapter we analyze the performance of the storage and retrieval algorithms. We investigate their load balancing performance as well as the resulting disk efficiency. We start with a probabilistic analysis of random redundant storage. With this analysis for the block-based approach we derive upper bounds on the probability that the maximum load is at least a certain value. In Section 5.2 we analyze with simulations the performance of the retrieval algorithms for RDS, partial duplication, and random striping with parameter $r = 2$. We compare the block-based and time-based retrieval algorithms regarding period length. We use the average period length, a 99% value of the observed values, and the worst-observed value in our comparison. In Section 5.3 we give additional comments on the performance of storage strategies.

5.1 Probabilistic analysis of block-based retrieval

In this section we give a probabilistic analysis of random redundant storage. With this analysis we show that random redundant storage in general, and random duplicate storage in particular, performs well, in the sense that the load is well distributed over the disks with high probability, where the load of a disk is defined as

the number of blocks assigned to it. We consider the following problem. Given are n requests that have to be retrieved from m disks, determine a bound on the probability that for an optimal load balance the maximum load is at least α for an integer value of $\alpha > \lceil \frac{n}{m} \rceil$. For a more elaborate probabilistic analysis of random redundant storage strategies we refer to Sanders, Egner & Korst [2000]. They show that random duplicate storage yields in each period a load of at most $\lceil \frac{n}{m} \rceil + 1$ with high probability for $n \to \infty$ and n/m fixed. Here, we are mainly interested in probabilistic bounds for practical values of n and m. We start this section with an analysis of random duplicate storage.

5.1.1 Duplicate storage

An instance of the retrieval problem for duplicate storage can be represented by an instance graph $G = (V, E)$ that was introduced in Chapter 3. The graph consists of a node for each disk, an edge between each pair of nodes, and a weight on each edge giving the number of blocks that has to be retrieved from one of the disks corresponding to the endpoints. Theorem 3.1 gives the relation between the optimal load distribution and the unavoidable load of a subset of the disks. We restate the result here. For duplicate storage an optimal distribution leads to a load of

$$l_{\max} = \max_{V' \subseteq V} \left\lceil \frac{1}{|V'|} \sum_{\{i,j\} \subseteq V'} w_{ij} \right\rceil . \tag{5.1}$$

This means that the probability of a certain load is related to the probability of the occurrence of a subset with a certain total weight. For completeness we state the following two propositions from probability theory that we use in our analysis [Motwani & Raghavan, 1995].

Proposition 1 [Principle of inclusion-exclusion]. Let $E_1, \ldots, E_N$ be arbitrary events. Then

$$\Pr\left[\bigcup_{i=1}^{n} E_i \right] = \sum_i \Pr[\, E_i \,] - \sum_{i<j} \Pr[\, E_i \cap E_j \,] + \sum_{i<j<k} \Pr[\, E_i \cap E_j \cap E_k \,] - \ldots \tag{5.2}$$

□

This proposition describes the probability of a union of events and holds for independent as well as dependent events. The next proposition states that a more simple form can be used to derive bounds. If we cut off the sum after an even number of

summands we get an upper bound and if we cut off the sum after an odd number of summands we get a lower bound.

Proposition 2 [Boole-Bonferonni inequalities]. Let $E_1, \ldots, E_N$ be arbitrary events. Then, for even k

$$\Pr\left[\bigcup_{i=1}^{n} E_i\right] \geq \sum_{j=1}^{k} (-1)^{j+1} \sum_{i_1<i_2<\ldots<i_j} \Pr\left[E_{i_1} \cap \ldots \cap E_{i_j}\right] \tag{5.3}$$

and for odd k

$$\Pr\left[\bigcup_{i=1}^{n} E_i\right] \leq \sum_{j=1}^{k} (-1)^{j+1} \sum_{i_1<i_2<\ldots<i_j} \Pr\left[E_{i_1} \cap \ldots \cap E_{i_j}\right]. \tag{5.4}$$

□

The goal is to find an upper bound on the probability that, for a given instance of the retrieval problem, an optimal assignment results in a maximum load of at least α. This means that we want to bound $P[l_{\max} \geq \alpha]$ from above. According to (5.1) we get

$$\begin{aligned} \Pr[\, l_{\max} \geq \alpha \,] &= \Pr\left[\max_{V' \subseteq V} \left[\frac{1}{|V'|} \sum_{\{i,j\} \subseteq V'} w_{ij}\right] \geq \alpha\right] \\ &= \Pr\left[\exists V' \subseteq V : \sum_{\{i,j\} \subseteq V'} w_{ij} \geq (\alpha - 1)|V'| + 1\right]. \end{aligned} \tag{5.5}$$

For a given a subset $V' \subseteq V$, we can determine the probability that it has an unavoidable load of at least α. We see this as an event, such that (5.5) is a union of events and we can apply the principle of inclusion-exclusion. As the exact computation of (5.2) is too complicated, we use a Boole-Bonferonni inequality with $k = 1$ to get an upper bound. We take only the first summand, as adding the next two summands makes the computation much more complex, whereas the accuracy will not be influenced that much. Later in this section we compare the upper bounds with simulation results and thereby show that the bound for $k = 1$ is sufficiently good for the values of α that we are interested in. The Boole-Bonferonni inequality gives

$$\Pr\left[\exists V' \subseteq V : \sum_{\{i,j\}\subseteq V'} w_{ij} \geq \alpha(|V'|-1)+1\right] \leq$$
$$\sum_{V'\subseteq V} \Pr\left[\sum_{\{i,j\}\subseteq V'} w_{ij} \geq \alpha(|V'|-1)+1\right]. \tag{5.6}$$

Each block is stored on a randomly chosen pair of disks. To generate a problem instance, we randomly choose an edge from the instance graph for each block. Whether a block contributes to the load of a subset V' can then be seen as a trial with success probability p, where

$$p = \frac{\#\text{ edges in } V'}{|E|} = \frac{\frac{1}{2}|V'|(|V'|-1)}{\frac{1}{2}m(m-1)}. \tag{5.7}$$

For a given subset V' the total load is the result of n independent trials with success probability p, such that the load of a subset V' is binomially $\mathcal{B}(n,p)$ distributed. This means that

$$\Pr\left[\sum_{\{i,j\}\subseteq V'} w_{ij} = k\right] = \binom{n}{k} p^k (1-p)^{n-k}. \tag{5.8}$$

For convenience, we define the probability that a $\mathcal{B}(n,p)$ distributed random variable is at least β as $F(n,p,\beta)$, i.e.

$$F(n,p,\beta) = \sum_{i=\beta}^{n} \binom{n}{i} p^i (1-p)^{n-i}. \tag{5.9}$$

Using this definition we get $\sum_{V'\subseteq V} F(n,p,(\alpha-1)|V'|+1)$ as an upper bound for $\Pr[\, l_{\max} \geq \alpha\,]$.

To compute the upper bound we still have to consider a large number of terms, as we sum over all subsets. However, for duplicate storage, subsets result in the same probability if they have the same number of nodes. We can use this symmetry to decrease the number of terms considerably. We determine for each subset-size i, $1 \leq i \leq m$, the success probability p_i and the number of times that such a subset occurs. The success probability p_i depends on the ratio between the number of edges in the subset and the number of edges in the complete graph, so $p_i = \frac{i(i-1)}{m(m-1)}$,

and the number of times a subset occurs equals $\binom{m}{i}$. Then, we get

$$\begin{aligned} \Pr[\, l_{\max} \geq \alpha \,] &\leq \sum_{i=1}^{m} \binom{m}{i} F(n, p_i, (\alpha-1)i+1) \\ &= \sum_{i=1}^{m} \binom{m}{i} \sum_{j=(\alpha-1)i+1}^{n} \binom{n}{j} (p_i)^j (1-p_i)^{n-j}. \end{aligned} \tag{5.10}$$

With this equation we can compute the upper bounds on the probabilities. Table 5.1 gives the results for duplicate storage for a disk array of 10 disks.

n	$\alpha = n/m+1$	$\alpha = n/m+2$	$\alpha = n/m+3$
50	$3.17 \cdot 10^{-1}$ ($1.88 \cdot 10^{-1}$)	$2.66 \cdot 10^{-6}$ (0)	$9.12 \cdot 10^{-11}$ (0)
100	$2.52 \cdot 10^{-2}$ ($2.31 \cdot 10^{-2}$)	$1.02 \cdot 10^{-7}$ (0)	$3.67 \cdot 10^{-13}$ (0)
150	$3.22 \cdot 10^{-3}$ ($3.16 \cdot 10^{-3}$)	$2.42 \cdot 10^{-8}$ (0)	$2.46 \cdot 10^{-14}$ (0)
200	$4.51 \cdot 10^{-4}$ ($3.6 \cdot 10^{-4}$)	$1.33 \cdot 10^{-8}$ (0)	$3.24 \cdot 10^{-15}$ (0)
250	$6.53 \cdot 10^{-5}$ ($8 \cdot 10^{-5}$)	$3.41 \cdot 10^{-9}$ (0)	$7.32 \cdot 10^{-16}$ (0)

Table 5.1. Upper bounds on the probability for three values of α and five values of n for a disk array of 10 disks. Within brackets BRP simulation results are included for comparison, based on experiments with 100,000 instances.

Table 5.1 shows that solving the block-based retrieval problem to optimality results in a perfect load balance with a probability over 97% in case of 100 block requests per period. For a smaller number of blocks this probability decreases, whereas for a larger number of blocks this probability becomes nearly 100%. Furthermore, we notice that the probabilities of a load of at least $n/m+2$ are negligibly small, even for 50 block requests. The values in this table are upper bounds on the actual probabilities. To validate the bounds we added simulation results for BRP. The values between brackets in Table 5.1 give the fraction of randomly generated instances that result in a maximum load that is at least α. Comparing the simulation results with the upper bounds on the probabilities we can conclude that the upper bounds are quite close to the actual probabilities, in particular for a larger number of requests per period.

It is worth mentioning that a major share of the upper bounds is generated by the large subsets. This is illustrated by Table 5.2, where the value of each of the terms of (5.10) is reported separately for different values of the subset-size i for 100 block requests and 10 disks, and $\alpha = 11$. Over 90% of the value of the upper bound is generated by subsets with 9 disks, and over 99% by subsets of 8 and 9 disks.

	$i=2$	$i=4$	$i=6$	$i=8$	$i=9$
$\alpha=11$	$3.25\cdot10^{-13}$	$1.53\cdot10^{-9}$	$2.92\cdot10^{-6}$	$1.78\cdot10^{-3}$	$2.33\cdot10^{-2}$

Table 5.2. Upper bounds on the probability for a fixed subset size for $\alpha=11$ and 100 requests on 10 disks.

Table 5.1 shows the probabilities for settings where m divides n. In case $\lceil n/m\rceil > n/m$ the probabilities that the maximum load is at least $\lceil n/m\rceil+1$ are smaller. To illustrate this we give in Table 5.3 the upper bounds on the probability that the load is at least 11 in case of a disk array of 10 disks and 92 up to 101 requests per period.

	$n=92$	$n=94$	$n=96$	$n=98$	$n=100$	$n=101$
$\alpha=11$	$1.98\cdot10^{-6}$	$3.19\cdot10^{-5}$	$4.58\cdot10^{-4}$	$4.20\cdot10^{-3}$	$2.52\cdot10^{-2}$	1

Table 5.3. Upper bounds on the probability for $\alpha=11$ for different numbers of requests on a disk array of 10 disks.

Figure 5.1 extends these results. It depicts the upper bounds on the probabilities that the optimal load for a disk array of 10 disks and 40 up to 100 requests is at least $\lceil n/m\rceil+1$. We see that the probability increases repeatedly towards the point that m divides n and than drops to values close to zero. In fact, they are less than 10^{-5}. This means that having some load balancing freedom, coming from the $m\lceil n/m\rceil-n$ 'empty' places in a schedule, decreases the probability of an overload considerably.

5.1.2 Partial duplication

We show in this section how the above analysis for duplicate storage can be adapted to partial duplication. We define q to be the fraction of the requested blocks that are stored twice. Consequently $1-q$ is the fraction of blocks stored only once, the so-called singly stored blocks. We redefine the unavoidable load as follows. In the instance graph we define the weight of a node as the number of singly stored blocks to be retrieved from the corresponding disk. Then, as the load is no longer only in the edges but also in the nodes, the total weight of a subset V' becomes the sum of the weights of the nodes in V' plus the weights of the edges with both nodes in V'. The unavoidable load is this total weight divided by the number of nodes in V'. With this definition we can prove an unavoidable load theorem for partial duplication following the proof of Theorem 3.1 and we can adapt the analysis of the previous section. Again the unavoidable load of a subset V' is a random variable

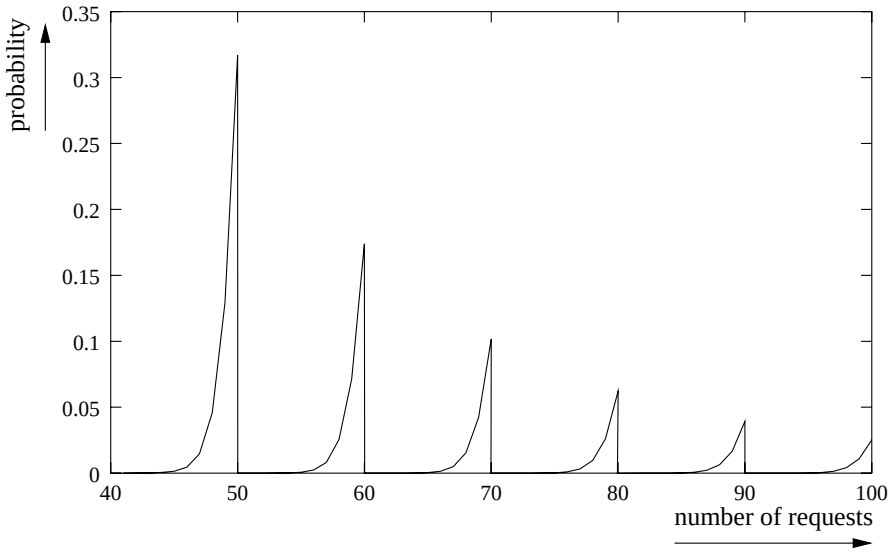


Figure 5.1. Upper bound on probability for $\alpha = \lceil n/m \rceil + 1$ for 40 up to 100 requests on 10 disks.

that is binomially $\mathcal{B}(n,p)$ distributed, where the probability p equals

$$p = q\frac{|V'|(|V'|-1)}{m(m-1)} + (1-q)\frac{|V'|}{m}. \tag{5.11}$$

Then, we use the same evaluation as described in the previous section to derive the results that are given in Table 5.3. We give in the table simulation results for $100,000$ instances for comparison.

Table 5.4 shows that the bound gives a good estimate for the tail of each distribution, if we compare the probabilistic results with the simulation results. If the fraction of duplicated blocks is small, the upper bound on the probability is larger than 1. This means that for these cases the estimation is too rough, which is a result of using the Boole-Bonferonni inequality with $k = 1$. However, we are mainly interested in the tails of the distributions and for these values the upper bounds give a good estimation of the actual probabilities.

Table 5.4 gives a good indication of the overall performance of fractional duplication, and it shows that the influence of duplication is really large. The values in the top row show the results for random single storage, where we see that approximately 10% of the instances results in a load of at least 18. The tail of the estimated distribution of random single storage is large and a maximum load of 25

q	$\alpha = 11$	$\alpha = 12$	$\alpha = 13$	$\alpha = 14$	$\alpha = 15$	$\alpha = 16$	$\alpha = 17$	$\alpha = 18$
0	1	1	1	1	1	$7.39 \cdot 10^{-1}$	$2.82 \cdot 10^{-1}$	$1.15 \cdot 10^{-1}$
	1	$9.99 \cdot 10^{-1}$	$9.68 \cdot 10^{-1}$	$8.28 \cdot 10^{-1}$	$5.95 \cdot 10^{-1}$	$3.63 \cdot 10^{-1}$	$1.99 \cdot 10^{-1}$	$9.99 \cdot 10^{-2}$
0.1	1	1	1	1	$6.92 \cdot 10^{-1}$	$2.39 \cdot 10^{-1}$	$9.18 \cdot 10^{-2}$	$3.64 \cdot 10^{-2}$
	1	$9.77 \cdot 10^{-1}$	$8.20 \cdot 10^{-1}$	$5.56 \cdot 10^{-1}$	$3.17 \cdot 10^{-1}$	$1.61 \cdot 10^{-1}$	$7.50 \cdot 10^{-2}$	$3.28 \cdot 10^{-2}$
0.2	1	1	1	$6.53 \cdot 10^{-1}$	$1.98 \cdot 10^{-1}$	$6.96 \cdot 10^{-2}$	$2.58 \cdot 10^{-2}$	$9.54 \cdot 10^{-3}$
	$9.98 \cdot 10^{-1}$	$8.40 \cdot 10^{-1}$	$5.24 \cdot 10^{-1}$	$2.72 \cdot 10^{-1}$	$1.26 \cdot 10^{-1}$	$5.34 \cdot 10^{-2}$	$2.15 \cdot 10^{-2}$	$8.11 \cdot 10^{-3}$
0.3	1	1	$6.48 \cdot 10^{-1}$	$1.59 \cdot 10^{-1}$	$4.97 \cdot 10^{-2}$	$1.69 \cdot 10^{-2}$	$5.79 \cdot 10^{-3}$	$1.91 \cdot 10^{-3}$
	$9.68 \cdot 10^{-1}$	$5.20 \cdot 10^{-1}$	$2.26 \cdot 10^{-1}$	$9.31 \cdot 10^{-2}$	$3.63 \cdot 10^{-2}$	$1.31 \cdot 10^{-2}$	$4.52 \cdot 10^{-3}$	$1.47 \cdot 10^{-3}$
0.4	1	$8.15 \cdot 10^{-1}$	$1.25 \cdot 10^{-1}$	$3.27 \cdot 10^{-2}$	$1.00 \cdot 10^{-2}$	$3.11 \cdot 10^{-3}$	$9.36 \cdot 10^{-4}$	$2.67 \cdot 10^{-4}$
	$8.22 \cdot 10^{-1}$	$2.01 \cdot 10^{-1}$	$6.51 \cdot 10^{-2}$	$2.17 \cdot 10^{-2}$	$6.59 \cdot 10^{-3}$	$1.78 \cdot 10^{-3}$	$5.10 \cdot 10^{-4}$	$1.30 \cdot 10^{-4}$
0.5	1	$1.16 \cdot 10^{-1}$	$1.95 \cdot 10^{-2}$	$5.08 \cdot 10^{-3}$	$1.39 \cdot 10^{-3}$	$3.73 \cdot 10^{-4}$	$9.03 \cdot 10^{-5}$	$2.24 \cdot 10^{-5}$
	$5.58 \cdot 10^{-1}$	$4.27 \cdot 10^{-2}$	$1.02 \cdot 10^{-2}$	$2.89 \cdot 10^{-3}$	$7.60 \cdot 10^{-4}$	$2.10 \cdot 10^{-4}$	$2.00 \cdot 10^{-5}$	0
0.6	$9.92 \cdot 10^{-1}$	$1.20 \cdot 10^{-2}$	$2.05 \cdot 10^{-3}$	$4.77 \cdot 10^{-4}$	$1.09 \cdot 10^{-4}$	$2.35 \cdot 10^{-5}$	$4.76 \cdot 10^{-6}$	$9.01 \cdot 10^{-7}$
	$3.18 \cdot 10^{-1}$	$5.02 \cdot 10^{-3}$	$8.70 \cdot 10^{-4}$	$2.20 \cdot 10^{-4}$	$5.00 \cdot 10^{-5}$	0	0	0
0.7	$3.20 \cdot 10^{-1}$	$7.27 \cdot 10^{-4}$	$1.03 \cdot 10^{-4}$	$1.88 \cdot 10^{-5}$	$3.26 \cdot 10^{-6}$	$5.28 \cdot 10^{-7}$	$7.98 \cdot 10^{-8}$	$1.12 \cdot 10^{-8}$
	$1.73 \cdot 10^{-1}$	$2.90 \cdot 10^{-4}$	$4.00 \cdot 10^{-5}$	0	0	0	0	0
0.8	$1.26 \cdot 10^{-1}$	$2.03 \cdot 10^{-5}$	$1.17 \cdot 10^{-6}$	$1.44 \cdot 10^{-7}$	$1.67 \cdot 10^{-8}$	$1.79 \cdot 10^{-9}$	$1.79 \cdot 10^{-10}$	$1.68 \cdot 10^{-11}$
	$8.85 \cdot 10^{-2}$	$1.00 \cdot 10^{-5}$	0	0	0	0	0	0
0.9	$5.53 \cdot 10^{-2}$	$9.87 \cdot 10^{-7}$	$3.55 \cdot 10^{-10}$	$1.98 \cdot 10^{-11}$	$1.14 \cdot 10^{-12}$	$6.08 \cdot 10^{-14}$	$3.02 \cdot 10^{-15}$	$1.40 \cdot 10^{-16}$
	$4.59 \cdot 10^{-2}$	0	0	0	0	0	0	0
1	$2.52 \cdot 10^{-2}$	$1.02 \cdot 10^{-7}$	$3.67 \cdot 10^{-13}$	$5.96 \cdot 10^{-18}$	$7.59 \cdot 10^{-22}$	$5.92 \cdot 10^{-25}$	$9.47 \cdot 10^{-28}$	$1.66 \cdot 10^{-33}$
	$2.31 \cdot 10^{-2}$	0	0	0	0	0	0	0

Table 5.4. Probabilistic results for partial duplication for 100 requests on 10 disks, where we depict vertically the fraction of blocks that is stored twice and horizontally several values of α. The upper value in each entry gives the upper bound on the probability and the lower value gives the result obtained by a simulation experiment.

blocks is still likely to occur. Increasing the fraction of duplicated blocks shows a large improvement in performance. The results for 0.8 and 0.9 are sufficiently good to be used in practice.

In Table 5.5 we show the trade-off between storage requirements and error probability. The table gives the number of requests that can be served per period by a disk array of 10 disks for a given fraction of duplication q and error probability.

The results show that for an error probability of 10^{-9} the fractional duplication strategies perform poorly compared to full duplication. For the smaller error probabilities the differences are smaller. Another trade-off that can be read from this table is storage requirements versus error probability. For example, we can retrieve 101 blocks with an error probability of 10^{-6} and full duplication, but also with an

	q										
error prob.	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
10^{-3}	35	38	43	48	55	66	81	101	104	106	106
10^{-6}	20	22	24	27	30	36	43	56	90	100	101
10^{-9}	14	15	16	17	19	22	26	32	46	87	96

Table 5.5. Number of requests that can be served per period by an array of 10 disks for a given fraction of duplication and error probability.

error probability of 10^{-3} and 70% of the blocks duplicated.

5.1.3 Random striping

In Section 3.1 we stated that the unavoidable load theorem also holds for BRP in case of random striping. We gave in that section the ILP formulation for random striping with parameter $r = 2$ and we defined the unavoidable load for that situation. Here we derive probabilistic results for random striping with $r = 2$.

If we consider a fixed subset I, with $|I| = i$, we can distinguish three possible situations for each block: (*i*) the block has all three disks in I, (*ii*) the block has two disks in I, and (*iii*) the block has no disks or one disk in I. The contribution to the total unavoidable load of I, measured in subblocks, is two, one, and zero, respectively. The total load within a subset of the disks is then multinomially distributed with the probability that a block has all three disks in I being

$$p_3 = \frac{\binom{i}{3}}{\binom{m}{3}}, \tag{5.12}$$

the probability that it has two disks in I being

$$p_2 = \frac{(m-i)\binom{i}{2}}{\binom{m}{3}}, \tag{5.13}$$

and the probability that it has one or zero disks in I given by

$$p_{01} = 1 - p_2 - p_3 \tag{5.14}$$

With these probabilities we can bound the probability that the minimum maximum load is at least α as follows.

$$\Pr[\, l_{\max} \geq \alpha \,] \leq \sum_{i=2}^{m-1} \binom{m}{i} f(m,n,\alpha,i), \tag{5.15}$$

where $f(m,n,\alpha,i)$ is the probability that an overload occurs in a given set I of size i. Using the definition of the multinomial distribution we get for $i = 3,\ldots,m$

$$f(m,n,\alpha,i) = \sum_{j=0}^{n} \sum_{\substack{k=\max\{0,\\ (\alpha-1)i+1-2j\}}}^{n-j} \frac{n!}{j!k!(n-j-k)!}(p_3)^j(p_2)^k(p_{01})^{n-j-k}, \qquad (5.16)$$

where j gives the number of blocks that contribute two to the unavoidable load and k the number of blocks that contribute one. To get the summation bounds in (5.16) we used that a subset of size i implies a load of at least α, if $2j+k \geq (\alpha-1)i+1$.

For $i = 2$, there are no blocks that contribute two to the total load such that

$$f(m,n,\alpha,2) = \sum_{k=(\alpha-1)2+1}^{n} \binom{n}{k}(p_2)^k(1-p_2)^{n-k}.$$

Table 5.6 gives upper bounds on the probability that the minimum maximum load is at least α, for $\alpha = 2n/m+1$, $2n/m+2$, and $2n/m+3$ and for $n = 50,\ldots,250$. The load of a disk is expressed as the number of subblocks assigned to that disk. In each entry we also give the value that resulted from simulation.

n	$\alpha = 2n/m+1$	$\alpha = 2n/m+2$	$\alpha = 2n/m+3$
50	$7.78\cdot10^{-1}$ ($3.60\cdot10^{-1}$)	$2.65\cdot10^{-4}$ ($4\cdot10^{-5}$)	$4.46\cdot10^{-8}$ (0)
100	$1.01\cdot10^{-1}$ ($8.64\cdot10^{-2}$)	$3.85\cdot10^{-5}$ ($4\cdot10^{-5}$)	$5.07\cdot10^{-9}$ (0)
150	$2.17\cdot10^{-2}$ ($2.12\cdot10^{-2}$)	$1.58\cdot10^{-3}$ ($3\cdot10^{-5}$)	$1.17\cdot10^{-9}$ (0)
200	$5.22\cdot10^{-3}$ ($5.14\cdot10^{-3}$)	$6.63\cdot10^{-6}$ (0)	$5.05\cdot10^{-10}$ (0)
250	$1.31\cdot10^{-3}$ ($1.23\cdot10^{-3}$)	$2.38\cdot10^{-6}$ (0)	$4.24\cdot10^{-10}$ (0)

Table 5.6. Upper bounds for BRP for random striping with $r = 2$ for three values of α and 50 up to 250 requests on 10 disks. Within brackets BRP simulation results are included for comparison, based on an experiment with 100,000 instances.

The results show that random striping with parameter $r = 2$ results in good load balancing, in the sense that the subblocks can be distributed over the disks in a balanced way. We see that the probability of a perfectly balanced load decreases compared to full duplication. However, we note that the load for random striping is measured in subblocks, such that an imbalance of one block is not the same as for full duplication.

5.2 Simulation experiments for time-based retrieval

In the previous section we showed that redundant storage performs well, in the sense that a good load balance of the number of blocks can be obtained with high probability. In this section we are going to evaluate the retrieval algorithms. In our simulation experiments we use a set of disk parameters that represents current hard disk technology. For this example disk we run simulations for a number of settings and analyze the results. We are interested in the worst-case performance of the retrieval algorithms. For that reason we analyze the period of a fully loaded system, i.e. we have to retrieve a block for the maximum number of streams in each period.

In the simulation experiments we take a fixed block size and use the period length as a measure for the performance of an algorithm. However, in the design of a system the block size and the period length are chosen in such a way that one block is large enough to offer video for the length of a worst-case period. This means that if the simulations show that a resulting period length for a fixed block size is smaller than the block size divided by the maximum consumption rate, the block size can be decreased, such that it corresponds to this resulting period length. A new simulation experiment with this new block size might result in an even smaller period length and so on, till the block size and the period length converge. We do not discuss this snowball effect in this chapter, but in the next chapter we discuss the design of a video server and we show the actual impact of this snowball effect on performance measures as the number of admitted clients.

We start with a description of the setting of our simulation experiments. We discuss the general setting and indicate in the subsequent sections if we deviate from this setting. First of all we define the block size to be 1 MB. We assume that the blocks of video data are stored on an array of m identical disks. The parameters of the switch time function are $c = 9.3$ ms and $s = 14.3$ ms. Each disk contains 15 zones. Table 5.7 gives the disk parameters for each zone: (*i*) zone number, (*ii*) size of the zone, expressed as a fraction the total capacity of a disk, and (*iii*) the transfer time for a block of 1MB.

For duplicate storage the data blocks are distributed over the disks in the following way. If one of the two copies is in the slowest zone, i.e. the innermost zone, of one disk, the other one is in the fastest, i.e. the outermost zone of the other disk. As the outermost zone is much larger than the innermost one, the copies of the blocks of zone 14 are also in zone 1. Continuing this we get a list of possible combinations of zones for the two copies. Using the sizes of the zones, we can compute for each

zone number	size of zone (fraction of disk size)	transfer time (in ms for 1MB)
1 (outermost)	0.141141	22
2	0.141141	23.3
3	0.063063	24.9
4	0.075075	25.8
5	0.090090	27
6	0.078078	28.4
7	0.036036	29.6
8	0.072072	31.6
9	0.048048	32.8
10	0.045045	34.6
11	0.045045	35.5
12	0.045045	38.7
13	0.033033	40.6
14	0.048048	42.6
15 (innermost)	0.039039	45.7

Table 5.7. Disk parameters.

entry in this list the probability that a requested block is stored on this combination of disks.

The set-up of each simulation is then as follows. We randomly generate 100,000 problem instances where n blocks have to be retrieved from m disks and run the load balancing algorithms on each instance. The measure that we use to compare the algorithms is period length, i.e. the time at which all disks have finished the retrieval of the assigned blocks. We compare the maximum observed period length, the 99% percentile on the observed period lengths, and the average period length. Furthermore, we can derive for a setting and an algorithm an estimation of the distribution of the period length, by splitting up the time axis in intervals and counting for each interval the number of instances that results in a period length that falls in that interval.

The load balancing algorithms that we compare are the maximum flow algorithm for BRP and the LP rounding, LP matching, and the list scheduling heuristic for TRP, as introduced in Chapters 3 and 4. For convenience we shortly restate the algorithms before discussing the simulation results.

Maximum flow algorithm (MF). The maximum flow algorithm solves BRP to optimality. It minimizes the maximum number of blocks assigned to one of the disks, without taking the transfer times into account. The implemented algorithm starts with a list scheduling solution to obtain an initial assignment. Each block is assigned to the disk with smallest current load. In case of equal load one of the disks is chosen randomly. The initial solution is used as an input to the maximum

		n				
		50	100	150	200	250
1 MB	LPR	1.73	1.38	1.25	1.19	1.15
	LPM	1.31	1.16	1.11	1.08	1.07
5 MB	LPR	1.22	1.11	1.07	1.06	1.04
	LPM	1.38	1.19	1.13	1.10	1.08

Table 5.8. Performance bounds for LP rounding and LP matching for 10 disks, blocks of 1 MB and 5 MB, and an increasing number of requests.

flow algorithm. The maximum flow algorithm improves this initial solution and finds an optimal solution regarding the number of blocks. Then, the algorithm computes the period length resulting from the assignment.

LP rounding (LPR). The LP rounding algorithm solves the LP relaxation in which the switch time is added to the processing time. To compute the actual cost, the algorithm rounds up the y-variables, which corresponds to adding the remaining part of the switch time for each preempted part. Table 5.8 gives, for two block sizes and an increasing number of clients, the values of the performance bound of LP rounding, which was derived in Theorem 4.6.

LP matching (LPM). The LP matching algorithm also starts with the solution of the LP relaxation and uses a matching approach to assign each preempted job to one disk. To improve this non-preempted solution we perform the postprocessing step where we allow preemption again. Table 5.8 gives values for the performance bound of LP rounding, which was derived in Theorem 4.7.

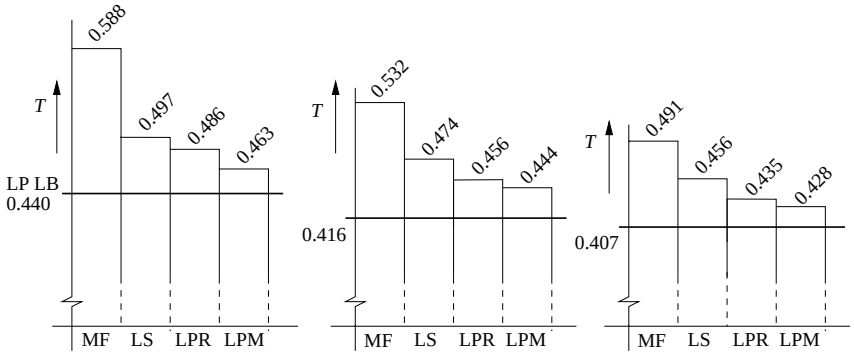
List scheduling heuristic (LS). The list scheduling heuristic starts with an empty assignment and assigns in each step a new block from the list of blocks to the disk with the smallest resulting load. In a second round we reconsider all jobs and check if a reassignment results in an improvement. Also in this algorithm we use the postprocessing step to improve this non-preempted solution.

For comparison we use the solution of the LP relaxation as a lower bound on the period length. Now we have set the general setting of the simulation experiment we continue with discussing the results. We start with an analysis of duplicate storage.

5.2.1 Duplicate storage

In this section we present the results for duplicate storage on an array of 10 disks. We compare the retrieval algorithms and quantify the performance improvement

due to the time-based approach. Figure 5.2 gives the maximum observed value, a 99% value and the average value for each of the algorithms for 100 requests.



(a) worst - case period length (b) 99% of the period length (c) average period length

Figure 5.2. Simulation results for 100 requests on 10 disks. The horizontal line gives the value of the lower bound.

Comparing the time-based retrieval algorithms we see that LPM outperforms the other two on all three measures. Comparing its average value with the average

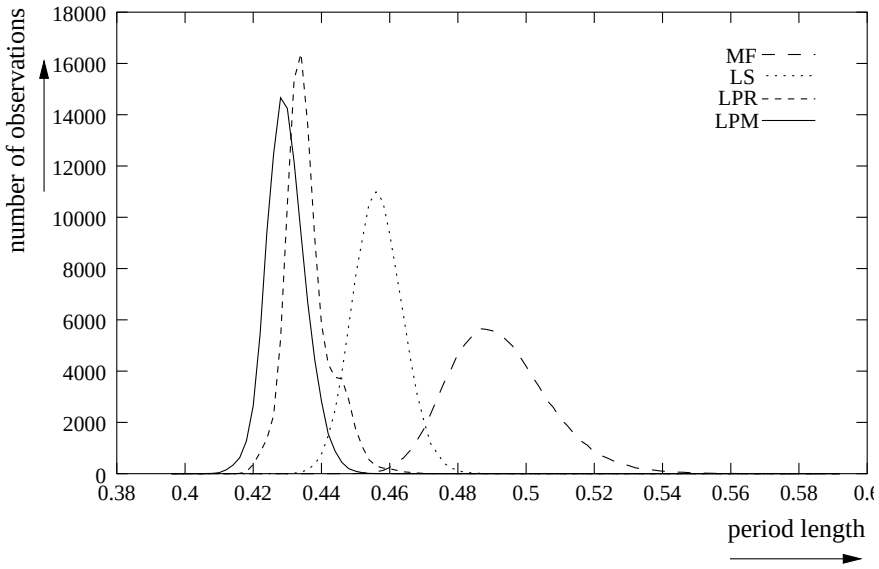


Figure 5.3. Estimated distribution of the period length for 100 requests on 10 disks.

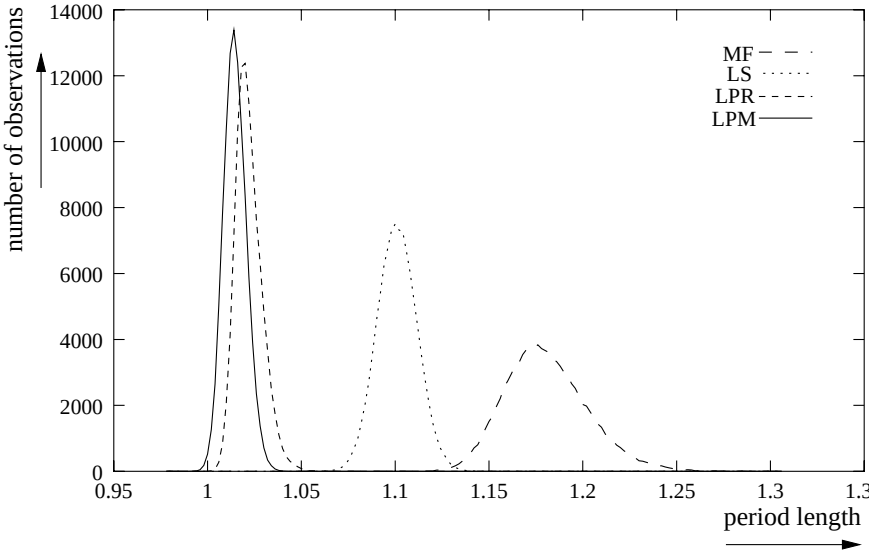


Figure 5.4. Estimated distribution of the period length for 250 requests on 10 disks.

period length of MF gives an improvement of 12.8%. The 99% value and the maximum observed value decrease with 16.5% and 21.3%, respectively. We even see that the worst-observed value for LPM is 5.7% smaller than the average value of MF. We can conclude that not only the average period length decreases by using the time-based approach instead of the block-based approach, but that also the variance is considerably lower. To illustrate this effect we show in Figure 5.3 the estimated distribution for each of the algorithms. As stated we derive this estimation by counting for each time interval the number of observations that fall in that interval.

We see in the figure that the graphs of both LP-based algorithms are much narrower than the MF graph. Furthermore we see that the very small tail of LPM just touches the beginning of the graph of MF. The graph of LS lies between the two. If we increase the number of blocks per period the observed effects are even stronger, as can be seen in Figure 5.4, where the graphs are depicted for 250 requests. For 250 blocks per period, the relative improvements of LPM compared to MF are 19.5%, 16.9%, and 14.2% for worst observed value, 99% value, and average value, respectively.

In Table 5.9 we give the results for duplicate storage for 50 up to 250 requests served by 10 disks. We present for each case the average value and the maximum observed value for each of the algorithms and the lower bound (LPLB).

	50 clients		100 clients		150 clients		200 clients		250 clients	
	avg.	max.	avg.	max.	avg.	max.	avg.	max.	avg.	max.
LPLB	0.212	0.252	0.407	0.440	0.602	0.631	0.797	0.821	0.992	1.021
MF	0.262	0.347	0.491	0.588	0.721	0.820	0.951	**1.059**	1.180	1.302
LS	0.244	0.286	0.456	0.497	0.670	0.710	0.885	0.926	1.100	1.146
LPR	0.240	0.286	0.435	0.486	0.630	0.675	0.826	0.869	1.021	1.070
LPM	0.235	0.275	0.429	0.463	0.623	0.655	0.818	0.851	1.014	**1.048**

Table 5.9. Average and maximum period lengths for different numbers of requests per period on 10 disks.

From the table we conclude that for this set of disk parameters LPM outperforms the other time-based algorithms. We also see that the difference between LPM and the lower bound decreases from 10% for 50 clients to less than 3% for 250 clients, on average as well as maximum observed. Another observation can be made by comparing the two entries printed in bold. It shows that the maximum observed value for LPM for 250 clients is smaller than the maximum observed value for MF for 200 clients. This means that a system that is configured on such a period length can serve approximately 25% more clients when using LPM rather than MF.

A disadvantage of synchronizing the disks in each period is that the disks that finish sooner have to wait till all disks are finished, before starting with the next batch. This is what we call idle time due to synchronization. We quantified this idle time for the LP matching algorithm and it turns out to be on average over all disks and all instances 0.78% of the period length. In case of the block-based approach it is 5.3%. This means that by using a time-based approach this disadvantage of synchronization becomes negligibly small.

Another way of evaluating the improvement of the time-based approach compared to the block-based approach is by looking at the fraction of blocks that was read from each zone during an entire simulation run. As the block-based max-flow algorithm does not take the transfer times into account we expect that for that case the fraction of each zone equals the percentage of the total disk capacity of that zone. Table 5.10 confirms this and shows the improvements of the time-based LP matching algorithm for $100,000$ iterations for 100 requests on 10 disks. Over 50% of the blocks is read from the fastest two zones, while they account for only 28% of the disk capacity. We also observe that the slower zones are almost never used. The average throughput of each disk while reading (excluding switch overhead) increases by exploiting the multi-zone property from 35.4 MB/s to 40.6 MB/s. This higher throughput can be used to increase the number of clients that can be served per disk. We quantify this effect on the number of admitted clients in Chapter 6.

zone	fraction of disk cap.$^{(i)}$	max-flow	LP matching$^{(ii)}$	improvement $\frac{(ii)}{(i)}$
1	0.141141	0.141273	0.278348	1.9721
2	0.141141	0.141264	0.260429	1.8451
3	0.063063	0.062953	0.101366	1.6074
4	0.075075	0.074947	0.104767	1.3955
5	0.090090	0.090202	0.099780	1.1076
6	0.078078	0.078042	0.066265	0.8487
7	0.036036	0.035999	0.023451	0.6508
8	0.072072	0.072076	0.032378	0.4492
9	0.048048	0.048062	0.013705	0.2852
10	0.045045	0.045054	0.007947	0.1764
11	0.045045	0.045080	0.005765	0.1268
12	0.045045	0.044921	0.002981	0.0662
13	0.033033	0.032966	0.001183	0.0358
14	0.048048	0.048033	0.001130	0.0235
15	0.039039	0.039123	0.000500	0.0128

Table 5.10. Fraction of blocks read from each zone for the block-based max-flow and the time-based LP matching algorithm for 100 requests on 10 disks. The last column shows the improvement of the time-based approach.

Table 5.11 gives similar results for 250 requests. The table shows that the improvements are even better. This can be explained by observing that each disk has to read 25 blocks on average, so more freedom is available for throughput optimization. The average throughput per disk while reading is in this case for the LP matching approach 41.2 MB/s, whereas we recall that the average throughput of a disk equals 35.4 MB/s.

We end this section with some further observations, regarding the performance of the algorithms in case the value of the slope of the switch time function is changed. If this slope, i.e. disk parameter s, is very small compared to the transfer times, the difference between LP rounding and the lower bound disappears, which means that LPR outperforms LPM in that case, as rounding is cheaper in that case than matching. This can also be seen in the performance bounds of Chapter 4, as ms becomes smaller than $p_{\max} + s$ for small values of s. If the switch slope is really large compared to the transfer times, LPR performs poorly, as rounding becomes very expensive. Also, the difference between the block-based and time-based approach becomes smaller in that case, as the number of blocks, and consequently the number of switches, forms the major part of the objective function. The improvement of the time-based approach compared to the block-based approach highly depends on the difference between the transfer times of the zones, and on the ratio between the slope of the switch time function and the transfer times.

zone	fraction of disk cap.$^{(i)}$	max-flow	LP matching$^{(ii)}$	improvement $\frac{(ii)}{(i)}$
1	0.141141	0.141116	0.282132	1.9989
2	0.141141	0.141236	0.279803	1.9824
3	0.063063	0.063099	0.117708	1.8665
4	0.075075	0.075098	0.123454	1.6444
5	0.090090	0.090110	0.107147	1.1893
6	0.078078	0.078059	0.057080	0.7311
7	0.036036	0.036076	0.014903	0.4136
8	0.072072	0.072027	0.013124	0.1821
9	0.048048	0.047977	0.003147	0.0655
10	0.045045	0.045084	0.000918	0.0204
11	0.045045	0.044979	0.000462	0.0103
12	0.045045	0.045074	0.000093	0.0021
13	0.033033	0.033008	0.000016	0.0005
14	0.048048	0.048004	0.000010	0.0002
15	0.039039	0.039052	0.000003	0.0001

Table 5.11. Fraction of blocks read from each zone for the block-based max-flow and the time-based LP matching algorithm for 250 requests on 10 disks. The last column shows the improvement of the time-based approach.

5.2.2 Partial duplication

We continue this chapter by analyzing the performance of the retrieval algorithms for partial duplication. In partial duplication a subset of the blocks is duplicated. The remaining blocks are stored once. We use q as the fraction of blocks that is duplicated. We start with evaluating this storage strategy as follows. We generate instances in which the number of duplicated blocks exactly equals qn. This means that we assume that we can control in each period the number of requested blocks that is stored twice. This can be done, e.g. by admission control in the following way. In case the number of running movies that is stored only once reaches $(1-q)n$, new clients are only offered duplicated movies. We compare these simulation results with a second experiment where each block of a generated instance is a duplicated block with probability q. In the latter experiment the total variation is larger.

The setting of the first simulation experiment is as follows. We use the general setting, but $(1-q)n$ of the requested blocks are stored only once. For these blocks we randomly select a zone, from which the transfer time can be determined. The probability that a block is stored in a certain zone equals the fraction of the disk capacity of that zone. In Figure 5.5 we compare LPM with MF for an increasing fraction of duplication.

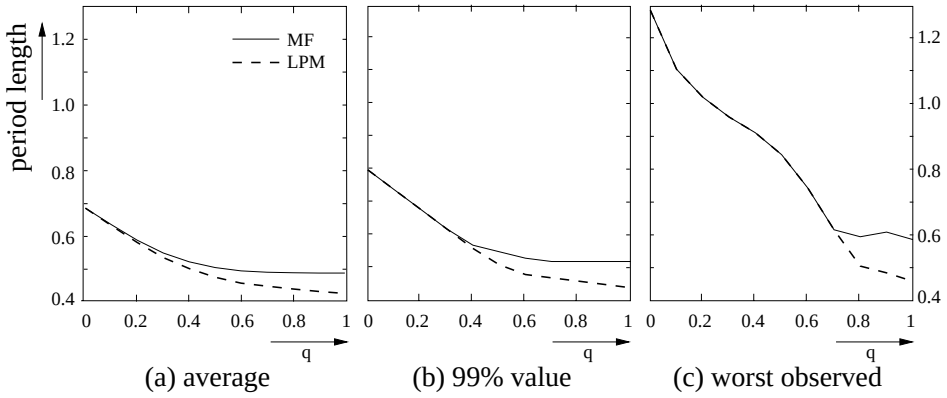


Figure 5.5. Average period length, 99% value of period lenght, and worst observed period length for MF and LPM for partial duplication with $q = 0, 0.1, \ldots, 1$ for 100 requests on 10 disks.

The simulations show that also for partial duplication LPM outperforms the other two time-based algorithms. We note that for the case $q = 0$ no scheduling decisions have to be made, such that there is no difference between the algorithms. The figure shows that if the fraction of duplicated blocks is small, almost no difference can be observed between the two algorithms. This can be explained as follows. If the disk that has the largest load has to retrieve only singly stored blocks, no scheduling algorithm can change this. We observe that all algorithms give the same worst-observed result, such that in these instances that is most likely to be the case. By increasing the fraction of duplication we see that the difference between the block-based and the time-based approach increases. To show the effect of increasing the fraction of duplication in a more detailed way we give in Figure 5.6 the estimated distribution of LPM for increasing values of q.

For the above results we assumed that the singly stored blocks were randomly stored over the zones. We can improve on this by storing the singly stored blocks in the middle zones. For the duplicated blocks we still assume that we have a fast and a slow copy. This means that if, for example, 60% of the requested blocks is stored twice, for each duplicated block one copy is stored on the outer 30% of one disk and one copy on the inner 30% of another, again in such a way that the blocks in the slowest zone have a copy in the fastest zone. The singly stored blocks are stored on the remaining 40% of the disks. Table 5.12 compares the simulation results for this centered partial duplicated storage strategy with the original strategy.

We see that the results of the MF as well as the LPM algorithm improve by us-

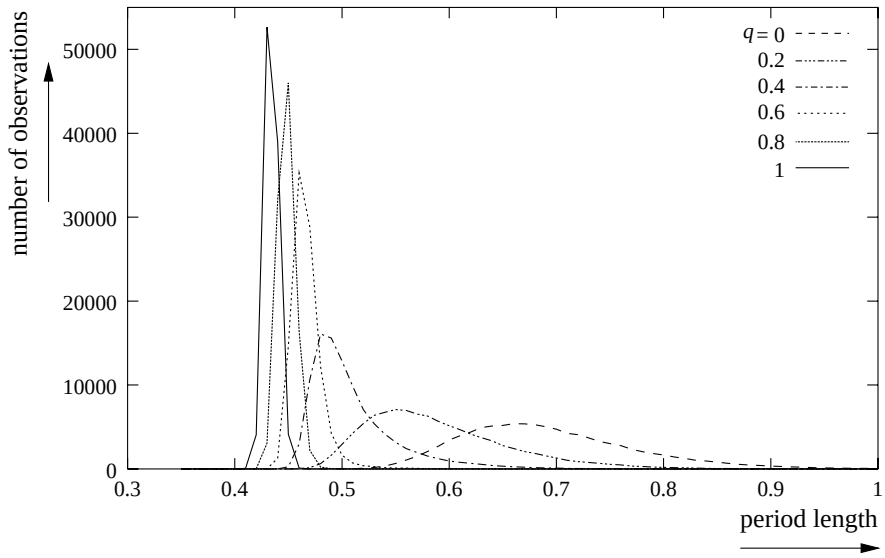


Figure 5.6. Estimated distribution for LPM for partial duplication for increasing values of q for 100 requests on 10 disks

	MF				LPM			
	original		centered		original		centered	
q	avg.	max.	avg.	max.	avg.	max.	avg.	max.
0.8	0.492	0.595	0.491	0.591	0.443	0.505	0.438	0.498
0.6	0.498	0.744	0.498	0.699	0.461	0.744	0.454	0.699
0.4	0.526	0.913	0.524	0.850	0.504	0.913	0.496	0.850
0.2	0.591	1.022	0.590	1.019	0.584	1.022	0.581	1.019

Table 5.12. Maximum observed and average value for the period length for the original and the centered partial duplication storage strategies for 100 requests on 10 disks.

ing the centered partial duplication, especially in the maximum observed value. However, the centered strategy is less flexible in the sense that if this strategy is implemented in a server a change in the fraction of duplication means that a large fraction of the data needs to be reordered.

All above results hold under the assumption that we can control the number of duplicated blocks per period. If we drop this assumption, we only know the fraction q of blocks that is stored twice. This gives an extra source of variation. To test the influence of this variation we change the simulation as follows. For each generated block request, there is a probability q that the block is stored twice. The variation

in the generated instances becomes larger, but Table 5.13 show that the average results are fairly close, and that the maximum observed value becomes only slightly larger. We note that for $q = 0.8$ the maximum observed value for the MF algorithm is coincidently better for non-fixed q. The 99% values happen to be the same for both strategies. We conclude that due to the large number of requests per disk the retrieval algorithms can deal well with the extra variation.

	MF				LPM			
	original		non-fixed q		original		non-fixed q	
q	avg.	max.	avg.	max.	avg.	max.	avg.	max.
0.8	0.492	0.595	0.492	0.583	0.443	0.505	0.442	0.520
0.6	0.498	0.744	0.498	0.754	0.461	0.744	0.461	0.754
0.4	0.526	0.913	0.525	0.961	0.504	0.913	0.502	0.961
0.2	0.591	1.022	0.593	1.072	0.584	1.022	0.584	1.072

Table 5.13. Maximum observed and average value for the period length if the fraction of duplicated blocks is not fixed, compared to the original results for 100 requests on 10 disks.

5.2.3 Random striping

Random striping can be used to decrease the storage requirements compared to duplicate storage. In that sense it is an alternative for partial duplication. In this section we discuss the performance of random striping with parameter $r = 2$. This means that each block is split up into two subblocks, that a parity block is computed, that the three subblocks are stored on three different randomly chosen disks, and that each combination of two of the subblocks is sufficient to reconstruct the original block.

In the simulation experiment we used the following storage strategy. We split up each disk in three equal-sized parts, the slowest, the middle, and the fastest part. Then, we use the following rule. We consider the slowest one-third of the disks from inside to outside and the other two parts from outside to inside. Then, we combine in the same way as we did for duplicate storage. We couple the slowest block positions in the slowest one-third to the fastest block position in the other two parts. Consequently, a block that is stored in the innermost zone has a copy in the outermost zone and in the fastest part of the middle one-third of the disk. For our example disk this means that we have the following combinations of zone numbers (1-3-15, 1-4-15, 1-4-14, ... ,3-8-8). For each generated request one of these combinations is drawn randomly according to the probabilities following from the capacities of the zones.

We compare the LP rounding and the list scheduling algorithm as described in Section 4.4, with the block-based maximum flow algorithm. Table 5.14 gives the average, worst observed and 99% value for 100 and 200 requests per period.

	100 requests			200 requests		
	MF	LS	LPR	MF	LS	LPR
max.	0.682	0.639	0.626	1.269	1.176	1.171
99%	0.657	0.612	0.612	1.255	1.157	1.157
avg.	0.624	0.588	0.589	1.222	1.136	1.137

Table 5.14. Simulation results for random striping for 100 and 200 requests on 10 disks.

The results show that the improvement of the time-based approach is approximately 6% for the average value and 7% for the 99% value for LS as well as LPR. This means that the improvement is smaller than in case of duplicated storage, which can be expected as the load balancing freedom is smaller, such that there is less room to increase the disk efficiency. However, the storage overhead is now only 50%. We see that the LP rounding algorithm performs approximately the same as the list scheduling heuristic, except for the maximum observed value. If we use a second reassignment run in the list scheduling algorithm, it even outperforms the LP rounding algorithm. In case of 200 blocks the average observed period length is then 1.134 and the maximum observed 1.165. The main reason for the poor performance of the LP rounding algorithm is that if three fractional values are found in the LP solution, one x-value has to be rounded up, as described in Section 4.4. The result is that the difference between LP rounding and the LP lower bound is larger than in case of RDS.

In the next chapter we further compare the results of random striping with parameter $r = 2$ and the list scheduling algorithm, with the duplicated storage strategies, and in particular with partial duplication.

5.3 Discussion

In this section we give additional comments on the performance issues that we discussed in this chapter. Furthermore, we discuss some performance issues of storage strategies that we did not discuss in detail in this thesis.

Computation times. For systems with a large number of clients and disks the LP algorithms demand large processing power, as in each period a large LP problem

has to be solved. If the computation times for the LP algorithms become too large, list scheduling becomes a good alternative. We saw that the performance of the list scheduling algorithms is good and these algorithms are always fast. Therefore list scheduling can be preferred in applications. This remark especially holds for random striping as the LP tableau is very large and the list scheduling heuristic performs almost as well as the LP rounding algorithm.

Randomization and redundancy. The results in this chapter show that combining randomization and redundancy results in good load balancing performance and efficient disk usage. Figure 5.5 shows that randomization alone is not sufficient. By increasing the fraction of duplicated blocks, we see that the period length almost halves at full duplication compared to random single storage in case of 100 blocks and 10 disks. For the worst-observed value of the period length we see that the improvement is almost 70%. For larger instances the ratios become even worse. On the other hand we can also evaluate the influence of randomness. in Section 3.3 we discussed random chained declustering, where the copy of each block is stored on its subsequent disk. The instance graph of this storage strategy is a cycle of disks. Aerts, Korst & Egner [2000] analyze the performance of block-based algorithms for random chained declustering and for other regular instance graphs, in which the number of edges increases. Recall that the instance graph of full duplication is a complete graph. The results in that paper show that a significant improvement can be reached by increasing the degree of randomness.

Round-robin striping. We did not discuss the performance of round-robin striping in this chapter. It is known that for highly predictable streams, round-robin striping outperforms full striping in disk efficiency as larger blocks can be read. However, even for these highly predictable streams Muntz, Santos & Berson [1998] show that random redundant storage outperforms round-robin striping. That result and the fact that we are mainly interested in strategies that are able to deal with variable bit-rates and unpredictable interactions are the reasons that we omitted round-robin striping in the evaluation.

Disk failures. When using a large number of disks, the probability of a failing disk is no longer negligible. This means that in the design of a server, disk failures should be taken into account. In Chapter 2 we introduced three storage strategies and explained how each strategy can deal with a failing disk. In the random redundant storage strategies the load of the failing disk is distributed over the others. In this thesis we did not take disk failures into account, but here we sketch how the probabilistic results of this chapter can be adapted to find upper bounds in case of disk failures for duplicate storage. If a disk fails, the weights of the edges adjacent to that disk shift towards the alternative disks. The result is that the number

of edges that generate load in a subset I that does not contain the failing disk, increases by $|I|$. This means that we get in case of duplicate storage and a disk failure the following definition for p instead of equation (5.7),

$$p_{\text{fail}} = \frac{\frac{1}{2}|I|(|I|-1)+|I|}{\frac{1}{2}m(m-1)} = \frac{\frac{1}{2}|I|(|I|+1)}{\frac{1}{2}m(m-1)}, \tag{5.17}$$

with $1 \leq |I| \leq m-1$. With this alternative definition of p we can follow the computation as explained in Section 5.1 to derive upper bounds on the probability of a certain load.

6

Server Design

In this chapter we evaluate the strategies and algorithms of the previous chapters in different system settings. The goal is to show what the effects are of the improvement of random redundant storage compared to the conventional strategy of full striping, and of the time-based approach compared to the block-based approach. We do not aim at covering the complete spectrum of system design in this chapter, but we try to illustrate effects and trends for several system settings and variations in system parameters, such as block size and number of clients.

For a discussion on the issues involved in system design we refer to Gemmell, Vin, Kandlur, Rangan & Rowe [1995]. Several other papers give a nice description of the implementation of a prototype, such as the following four papers. Berenbrink, Brinkmann & Scheideler [1999] describe the hardware structure and the data placement strategy of the PRESTO multimedia server and give simulation results. Ghandeharizadeh & Muntz [1998] discuss the performance of a multimedia server named MITRA. Next to explaining the design of the prototype, they discuss several issues such as multi-zone disks, batching strategies to reduce bandwidth requirements, and VCR functionality. Muntz, Santos & Berson [1998] introduce the RIO multimedia server. The paper explains the working of the server, discusses the storage strategy, and presents probabilistic and simulation results. Shenoy, Goyal,

Rao & Vin [1998] discuss the implementation of a multimedia server called Symphony. The system supports both real-time and non-real-time requests and enables multiple block sizes. The paper also discusses the performance of the prototype and failure recovery in case one of the disks breaks down.

This chapter is organized as follows. In Section 6.1 we discuss the general setting of the cases that we analyze and we introduce the parameters and trade-offs that play a role the system design. In the two subsequent sections we discuss specific cases. In Section 6.2 we discuss the design of a video-on-demand server in an airplane or hotel and on a larger scale in, for example, a city or district. In Section 6.3 the focus is on professional applications, such as film editing, medical servers, or digital libraries. An important difference between these professional applications and video on demand is that in these applications the clients have a very active role, in the sense that they are browsing through the available data, instead of watching one video for a long time. Consequently, for these applications the response times a critical issue. Furthermore, for some of the applications, such as film editing, the bandwidth requirements of streams are typically much higher, and the requests are write as well as read requests. In Section 6.4 we present conclusions and possible extensions.

6.1 Case study introduction

Discussing the design of a server we mainly focus on the choice of which storage and retrieval strategy to use for a given set of system requirements. Performance aspects that are important in the design of such a system are, for example, response time and system cost. The system performance is influenced by the setting of the system, such as the storage and retrieval strategy that is used, the number of hard disks, and the bandwidth of the streams. We can distinguish a large number of parameters that influence the performance of a multimedia server. These parameters can take a role as a requirement in one setting and as a performance criterion in another. We start by indicating which parameters can be distinguished in Table 6.1.

If we want to state a problem definition it is necessary to assume fixed values for a subset of the parameters. The other parameters can then be used to optimize the system with respect to one or more performance criteria. The most obvious criteria are response time and cost per client.

Response time. From a client's point of view the response time is the time between the request for a media object and the actual start of playout at the client's terminal. However, we focus on the video server and do not take communication delays

disk	storage capacity (per zone)
	transfer rate (per zone) and switch time
	disk cost
disk array	number of disks
	storage and retrieval strategy
clients	number of clients
	maximum consumption rate
buffers	buffer strategy
	block size
	buffer cost

Table 6.1. Parameters in the design of a multimedia server.

into account, so we define the response time to be the time between the arrival of the request at the server and the start of sending out the video into the external network. The response time depends on the storage and retrieval strategy, the buffer strategy and and the block size. Response times can be considered both from a worst-case and average-case perspective. We use the worst-case response time as a performance criterion. As we use synchronized disks and triple buffering, this worst-case response time equals two times the period length.

System costs (cost per client). The variable costs of a multimedia server mainly consist of the cost of RAM and the cost of the hard disks of the disk array. So the cost per client depends on the block size and buffer strategy, on the number of disks, and also on the maximum number of admissible clients.

Next to these optimization criteria we can also use one of the above system parameters, for example, minimizing the number of disks. In the remainder of this chapter we use the same example disk as in the previous chapter, thereby fixing the transfer rate and storage capacity per zone and the parameters of the switch time function. The total storage capacity of the disk equals 40 GB. Furthermore, we assume that triple buffering is used as buffer strategy, so that within the server a buffer of size three times the block size is used for each stream. In the examples in this chapter we evaluate the consequences of variations in number of disks, stream bandwidth, block size, and number of clients on the choice for the storage and retrieval strategy.

The next two sections are organized as follows. We start each section with an explanation of the characteristics of the case and describe some possible applications. Then, we analyze several settings quantitatively and describe the results. We end both sections with some conclusions.

6.2 Video on demand

A video-on-demand server offers video streams to multiple clients simultaneously. As clients are expected to watch a video for a long time, the response times are not a critical issue. In fact, they can often be masked by a leader or advertisement. Examples of possible video-on-demand settings are the following.

- A hotel manager wants video on demand in her hotel, consisting of 200 rooms. She requests the response time to be smaller than 10 seconds and that at least 500 movies are offered. The question is to design a server at minimum cost that satisfies these requirements.
- An airline company wants to offer video on demand in its planes. It is very likely that all passengers of a plane want to see a movie simultaneously, so the number of clients equals the number of chairs in the plane. The number of movies does not need to be larger than 20, as in this case the video-on-demand system is an alternative to broadcasting a small number of movies. The question is to design a system with a minimal number of disks that enables all passengers to watch these movies on demand.
- A content provider wants to offer television on demand in a district of a town. The data is extracted from broadcast channels and consists of all television programs of the last week. The number of possible clients is very large, but the provider accepts an admission control algorithm that bounds the number of streams that are admitted simultaneously.

The examples give an idea of the broad range of video-on-demand applications. They also show that the requirements and optimization criteria can change per setting. To get an insight in the effect of changes of the parameters on the preferred storage strategy, we describe several settings and discuss some trade-offs in the remainder of this section.

6.2.1 Fixed number of disks

We start with the following scenario. Suppose that we have a disk array of ten disks, which we want to use for a video-on-demand server. We first assume that we still have the possibility to adapt the size of the total buffer space in the server. For this setting we maximize the number of clients that can be served with a fixed block size. We compare the results of striping, random striping with parameter $r = 2$ and LP matching (RS(2)), partial duplication with $q = 0.5$ and LP matching, random duplicate storage with max-flow (RDS-MF), and RDS with LP matching (RDS-LPM). Table 6.2 presents the results for a block size of 1 MB, 2 MB, and 5 MB. It gives the maximum number of clients for which the 99% value of the

period length is smaller than the period length corresponding to the block size, given that a client has a maximum bit rate of 6 Mb/s = 0.75 MB/s.

	striping	RS(2)	partial	RDS-MF	RDS-LPM
1 MB	75	230	300	274	330
2 MB	129	323	365	327	**408**
5 MB	**222**	414	420	370	472

Table 6.2. Maximum number of clients that can be offered simultaneously by 10 disks for a given block size.

The results show that for this setting the redundant storage strategies outperform full striping considerably, mainly due to the large switch overhead when using striping. In case of blocks of 1 MB, the striping subblocks are of size 0.1 MB, implying that the switch time is a factor two to five larger than the transfer time. For larger blocks we see that the efficiency of striping increases. Comparing the redundant data strategies we see that RDS with time-based load balancing enables the admission of the largest number of clients and that using the time-based retrieval algorithm enables 20%–28% more clients compared to the block-based approach. Furthermore, we see that the block-based approach for RDS outperforms random striping in case of a small block size, but that it is the other way around if the block size increases. This can be explained as follows. Random striping with LP matching exploits the multi-zone character of the disks, such that the disks are more efficiently, but for small blocks this effect does not compensate the larger switch overhead compared to RDS. Comparing random striping and partial duplication, both having storage overhead of 50%, we see that for smaller block sizes partial duplication outperforms random striping, but for larger block size random striping comes close to partial duplication. We can explain this as follows. For smaller blocks random striping loses performance because of a larger switch overhead. However, random striping can better exploit the multi-zone character of the disks, as in partial duplication for some blocks no alternative is available. Apparently, for larger blocks this effect compensates for the larger switch overhead.

The worst-case response time of the above settings does not depend on the storage strategy, but only on the block size and is 2.67, 5.33, and 11.33 seconds, respectively. The number of movies that can be stored on the array of 10 disks depends on the degree of duplication, and is 89 for striping, 59 for random striping and partial duplication, and 44 for RDS for movies of 100 minutes. The buffer size per client equals three times the block size, so the total buffer size is linearly dependent on the number of admitted clients.

Another point of view on the comparison between RDS and striping, is given by the following example. Consider in Table 6.2 the two entries printed in bold. By using RDS with LP matching the server with ten disks and a buffer of 2448 MB can admit 408 clients when using blocks of 2 MB. A server with ten disks and a buffer of 3330 MB that uses striping can still serve only 222 clients. Furthermore, the response time for the server with striping is larger than for the server with RDS.

6.2.2 Fixed number of clients

Consider the following design problem: Given a number of clients and a requirement on the response time, design a server with a minimum number of disks. We assume that the maximum response time should be at most 10 seconds and consider the problem for 100, 250, and 1000 clients. We configure the system such that the size of a block corresponds to the 99% value of the period length, which should be at most 5 seconds, to obtain a worst-case response time of at most 10 seconds. This means that the blocks should be at most 3.75 MB. For this block size we determine the minimum number of disks for which the period length is at most 5 seconds. Given this minimal number of disks, we minimize as a second criterion the response time, by decreasing the block size in steps of 0.25 MB. We consider the Table 6.3 gives the results for full striping, random striping with $r = 2$ and the list scheduling heuristic, partial duplication with $q = 0.5$ and LP matching, and RDS with LP matching.

	100 clients	250 clients	1000 clients
striping	4 (3 MB)	22 (3.75 MB)	infeasible
RS(2)	3 (2.5 MB)	7 (3 MB)	27 (3.75 MB)
partial	3 (2 MB)	7 (2 MB)	26 (3.75 MB)
RDS-LPM	3 (1.25 MB)	6 (2.5 MB)	22 (3.75 MB)

Table 6.3. Minimum number of disks needed to serve a given number of clients with a maximal response time of 10 seconds. Within brackets the minimum block size is shown for the determined number of disks.

The results show that full striping is not suited for large systems as can be expected, as the subblocks become too small and the required bandwidth cannot be reached. This is the case for 1000 clients. The redundant data storage strategies are competitive. We see that RDS outperforms the other two strategies, but we remark that the LP matching algorithm is really time consuming for large instances. This means that for the instances with 1000 clients we should switch to the list scheduling heuristic that was introduced in Chapter 4. If we would have used list scheduling

in case of RDS we would have needed 25 disks for 1000 clients, which means that random striping performs almost as good. For partial duplication the same remark holds, such that random striping outperforms partial duplication in case of a large number of clients

We give some final comments on the results of Table 6.3. If 60% of the data would have been duplicated in the partial duplication strategy, 6 disks would have been sufficient to serve 250 clients and in case 80% was duplicated 23 disks would have been sufficient for 1000 clients. The decrease in block size that is reported in the table results in a decrease in worst-case response time. This worst-case response time can be determined by multiplying the block size that is reported within brackets by a factor 2.67.

Note that the costs of hard disks and buffer have decreased dramatically in the past decade, such that the total costs become very low. For example, as the price of a hard disk of 40 GB is approximately € 100, using 22 disks for 1000 clients results in a disk cost per client of just over € 2. However, minimizing the number of disks to serve a given number of clients is also of advantage for the probability of disk failures and for simplicity within the server, as, for example, the internal network can be simpler.

6.2.3 Conclusion

The two scenarios discussed above show that duplication outperforms the alternatives in case of video on demand. The only drawback is the smaller number of movies that can be offered. Especially for smaller systems, with fewer disks, this might be a significant drawback. For these settings partial duplication and random striping offer interesting alternatives. Full striping is only competitive if the number of disks is small and the blocks large, where the latter means that the response time is high. In a striping strategy it is harder to exploit the multizone character of the disks, compared to the other strategies. without losing the independence between subsequently requested blocks. We also note that if information about the popularity of the movies is available, this can be used to improve partial duplication, such that this strategy becomes more competitive to full duplication. Finally, we note that over the past decades the storage performance of hard disks increased at a higher rate than the disk bandwidth. If the future development of hard disks follows these lines, redundant data strategies become even more preferable in the future.

6.3 Professional applications

In the video-on-demand applications of the previous section a client typically starts a video and spends a long time watching it. Now, we consider databases that contain video data that is used for browsing and editing. All the time clients send requests for (short) video files to the server. They browse through the data, so the most important performance criterion is response time. Below, we give some examples.

- A data agency gathers news clips from all over the world and offers news on demand to press agencies. A large number of incoming streams and a large number of outgoing streams should be combined. Typically, a small amount of the data is requested by a large number of clients. The clients want to browse through the video data to compose their own news reports.
- In a film editing studio, movies are constructed out of raw film material. The streams in such an environment have very high bandwidth requirements. Typically, the rate can be as high as the transfer rate of a disk. The request pattern is unpredictable. Over time, the editors request new streams, and they sometimes write a stream to disk. To create a good working environment, low response times are required.
- In a hospital a large database of short high-quality video files is available to the staff. The database contains for example X-ray videos. Upon request a doctor wants to see a certain file. Again response time is the main performance criterion.

6.3.1 Increasing bit-rates

In the previous section the bit-rate of the videos was assumed to be 6 Mb/s. In this section we evaluate the performance of the algorithms for streams with higher bit-rates. We first show how to deal with an increasing bandwidth using the results of the previous section. Then, we analyze the performance of the storage and retrieval strategies for bit-rates that are approximately as large as the disk bandwidth.

We first readdress the results presented in Table 6.2. The table gives the number of clients that can be served for a certain block size for several storage strategies. The data in the table can also be interpreted as the number of blocks that can be retrieved in a period, where the period length corresponds to the block size. This means that instead of retrieving one block per client per period, we can also retrieve two blocks per client per period, thereby serving half of the number of clients at a doubled bit rate. Note that the buffer size per client needs to be increased to avoid buffer underflow and overflow. To be more precise, in case at most two blocks

are read per client per period, a buffer size of five blocks is sufficient. To avoid underflow and overflow the buffer should request one block if the buffer filling at the beginning of the period is between three and four blocks and two blocks is the filling is at most three blocks. The worst-case response time remains two times the worst-case period length. So, using the last column of Table 6.2, we can conclude that a server with 10 disks and 1 MB blocks, can serve 330 clients at 6 Mb/s, and 165 clients at 12 Mb/s. At the cost of an even larger buffer per client 41 clients can be served at 48 Mb/s. The worst-case response time equals 2.67 s. This response time might be too large for browsing applications. This response time can be halved by using blocks of 0.5 MB. Then, 235 blocks can be retrieved per period if RDS with LP matching is used.

We continue with systems where the bit-rate of the requested streams is as large as the bandwidth of a disk. The bandwidth of the disk that we use in the simulations ranges from 22 to 45 MB/s. We evaluate the performance of systems that offer streams with a homogeneous bit-rate, ranging from 20 to 80 MB/s. As we are discussing browsing applications, the worst-case response time should be small, we assume one second, so the period length should be at most 0.5 s. This means that each client should receive in each period an amount of 10 to 40 MB of data. Table 6.4 gives the number of clients that can be served by 10 disks for five storage and retrieval algorithms for bit-rates ranging from 20 to 80 MB/s. The block size is not fixed, but is chosen in such a way that a maximum number of clients can be served. For the random redundant storage strategies there is a trade-off between the block size and the load balancing performance. If the block size increases, the switch overhead decreases, but also the number of blocks per period decreases. The latter effect restricts the possibilities for load balancing and exploiting the multi-zone character of the disks.

	block size	20 MB/s	40 MB/s	60 MB/s	80 MB/s
striping	variable	9	5	3	2
partial	2 MB	9	4	3	2
RS(2)	5 MB	12	6	4	3
RDS-MF	2 MB	10	5	3	2
RDS-LPR	3.33 MB	14	7	4	3

Table 6.4. Number of clients that can be served by ten disks for four possible bit-rates and five storage and retrieval strategies. The column with block sizes gives and optimal block size in the sense that the number of clients is maximized. For the random redundant storage strategies it is constant, but for striping the optimal block size is the amount of data to be received by each client in each period.

Looking at the results we see that for instances with very high bit-rates striping is competitive with the random redundant storage strategies. For the time-based approach of RDS, we used LP rounding to solve the retrieval problem as LP rounding outperforms LP matching for these large block sizes. RDS with LP rounding outperforms the other strategies. Partial duplication with 50% of the blocks stored twice performs poorly. The number of blocks per period becomes too small, such that there is not enough load balancing freedom available to obtain efficient disk usage. For random striping we used the LP rounding algorithm as this preemptive algorithm outperforms the list scheduling algorithm for these large blocks. For the redundant storage strategies the table gives the optimal block size. Increasing the block size any further results in a drop in performance, due to too little load balancing freedom. Striping uses blocks as large as the amount of data that a client needs per period, i.e. 10, 20, 30, 40 MB, respectively. In that way striping is able to exploit the possibility to increase the block size to the fullest. An idea to improve the performance of striping for these instances is to store the striped data only on the outer half of the disk. In that way the total amount of stored data is still the same as in case of RDS. Experiments show that such a system could serve 12, 7, 5, and 4 clients, respectively, so it outperforms RDS with LP rounding for the highest two bit-rates.

6.3.2 Reading versus writing

Until now we focussed on servers from which clients retrieve data. Storing the data on the server is assumed to be done off-line. However, if we consider film editing, a large fraction of the requests are write requests. In case of writing, redundancy results in an increase of the workload compared to non-redundant storage. In this section we discuss the effect of redundancy in servers that support write and read requests.

We again look at a server that contains MPEG streams, so we assume that all read and write requests concern streams of 0.75 MB/s. We use blocks of 0.5 MB and 2 MB so the worst-case response time equals 1.33 s and 5.33 s, respectively. In this analysis we apply the following algorithm for writing a duplicated stream. For each block two disks are chosen randomly and a combination of zones is chosen randomly in the same way as in the previously applied storage strategies. Then, both blocks are assigned. For random striping writing is done in a similar way. A more sophisticated writing algorithm would be needed to keep the preferred distribution of the data, but this is considered to be outside the scope of this thesis. Table 6.5 gives the results for four storage and retrieval strategies, two block sizes, and 33% and 50% write requests.

	write req.	striping	partial	RS(2)	RDS-LPM
0.5 MB	33%	50	129	120	162
	50%	50	116	108	118
2 MB	33%	129	258	255	270
	50%	129	224	230	226

Table 6.5. Number of streams that can be served by ten disks for 33% and 50% write requests for four storage and retrieval strategies and two block sizes.

As can be expected, the results show that RDS has the largest drop in performance when the fraction of write requests increases. We see that for 2 MB sized blocks and 50% writing the three random redundant storage strategies perform equally well. The small differences are not significant as the numbers are simulation results. It is worth mentioning that for this setting the variation in period length was smallest for random striping, which makes that strategy preferable. The smaller variation can be explained as follows. For RDS we see instances where the value of LP matching equals the lower bound, which is probably due to the fact that the disk with maximal load has only write requests. Such instances make that the worst-observed value is considerably larger than for random striping. For partial duplication a similar argument holds.

Combining the results of Table 6.4 and the fact that the performance of the redundant strategies decreases due to write requests, we can conclude that full striping outperforms the random redundant strategies in case of high bit-rates and a large fraction of write requests.

We end this section with a remark on the buffer strategy. For reading we assigned a single buffer of size three times the block size to each stream. In the experiments above we assumed that it is possible to control the fraction of write streams exactly, in such a way that in each period 50% of the blocks have to be written. Then, a buffer of three times the block size is sufficient. However, in practice the fraction of write requests varies over time. So a better buffer implementation is to have a large writing buffer, as it is not relevant from which client a block comes that has to be written. In this way the server can deal with short-term variations in the fraction of write requests. A detailed discussion of the buffer effects is considered outside the scope of the thesis.

6.3.3 Conclusion

The results of this section show that the redundant storage strategies perform well for high bit-rates. However, we saw that for small systems that offer streams with

very high bit-rates striping is at least competitive. Striping is even more preferable when the fraction of write requests increases. Furthermore, if bandwidth instead of storage is the bottleneck resource, striping over the outer half of the disk further improves this strategy.

For servers that combine reading and writing of MPEG streams we saw that if a small response time is required and the fraction of writing is 0.33, RDS is the best strategy. For a larger fraction of writing, random striping and partial duplication become at least competitive. Striping is not able to compete with the redundant strategies for MPEG streams when a small response time is required.

6.4 Discussion

In this chapter we evaluated the performance of random redundant storage strategies and the retrieval algorithms in the design of video-on-demand and other multimedia servers. The results show that random redundant storage is applicable and preferable in a wide range of settings. Furthermore, this chapter shows that a large number of trade-offs have to be considered in the design of a system. We highlighted several of them, but we do not aim to be complete in this matter. In this discussion section we describe related issues and add further comments.

Buffer size. The buffer requirement for a stream with a high bit-rate is large, as in each period a large amount of data arrives. A way to decrease this buffer requirement is by taking a smaller period and serving each client more frequently with smaller data blocks. That solution results in a larger switch overhead, so there is a trade-off between disk efficiency and buffer requirement. Another way to decrease the buffer requirement is by retrieving multiple blocks per period per stream, as explained in Section 6.3.1. If each client receives two half-sized blocks per period the buffer size needs to be 2.5 block sizes instead of 3. Furthermore, as more blocks have to be retrieved per period, the load balancing algorithms perform better. Again, this is at the cost of a larger switch overhead.

Performance guarantee. To configure the server we used in this chapter the 99%-value of the estimation of the period length that follows from simulations. This might lead to a period longer than the one that is accounted for, once every 100 periods, but note that it means that this is expected to happen once every 100 *worst-case* periods. A system that is configured on this value performs in practice much better, as, for example, there is only a very small probability that all clients require a maximum bit-rate at the same time. Next to that, one may alternatively use the 99.9% value or add a safety margin to the 99% value. The cases described in

this chapter illustrate how the techniques that are described in this thesis can be implemented and exploited. A performance evaluation in a prototype with actual streams can be a next step in the design of a server. In that way the actual failure probabilities can be measured.

Disk failures. An important performance issue of a video server is its performance when one disk fails. For RDS and random striping, all data is still available in case of a disk failure. For striping and partial duplication extra precautions should be taken to prevent loss of data. For wide striping it is possible to add one extra disk on which a parity subblock is stored for each block [Patterson, Gibson & Katz, 1988]. For partial duplication a similar parity method can be designed.

Heterogeneous streams. In the discussion section of Chapter 4 we explained that the time-based models can be used for heterogeneous streams. One way to do this is by splitting up the higher bit-rate videos into larger blocks. However, this leads to a more complex storage structure. Another way to deal with heterogeneous streams is by retrieving more blocks for clients that are watching at a higher bit-rate, in the same way as explained in the first part of Section 6.3.1. A minor disadvantage is that a more complex buffer algorithm is needed to avoid underflow and overflow for the clients that watch at higher bit-rates.

7

Conclusion

In this thesis we discussed the use of random redundant storage strategies in video-on-demand systems. In these strategies each video file is split up into blocks and each block is stored on one or more randomly chosen disks. We assume that the disks within the server are synchronized and that the server works in periods of variable length. In each period a large number of blocks has to be retrieved from the disks and the next period starts as soon as all disks have finished. The performance of a storage strategy is measured by the period length and this period length depends on the load balancing performance of a storage strategy and how efficient the disks are used.

We defined the retrieval problem for the random redundant storage strategies as follows. For each requested block it has to be decided which disk(s) to use for its retrieval such that the period length is minimized. We analyzed two versions of the retrieval problem. In the block-based retrieval problem (BRP) the period length is determined by the maximum number of blocks assigned to any disk. In the time-based retrieval problem (TRP) the period length is defined as the completion time of the disk that finishes last, where the switch times and the actual transfer times of the blocks are taken into account.

We modeled and analyzed the retrieval problems from a combinatorial optimization point of view. We showed that they form a special class of multiprocessor scheduling problems. We modeled BRP as an integer linear programming problem and we defined the edge weight partition problem as the special variant of BRP in case of duplicate storage. Furthermore, we related the problem to the maximum density subgraph problem and showed that BRP can be seen as a special case of the maximum flow problem. We modeled TRP as a mixed integer linear programming problem and showed that the model can be applied to a broad range of settings.

We proved that BRP can be solved in polynomial time, by showing that BRP is a special case of the maximum flow problem. TRP is proved to be NP-complete in the strong sense. We also discussed the complexity of some special cases. TRP with duplicate storage and no preemption and set-up times is NP-complete in the strong sense. TRP with the number of machines defined in the problem definition can be solved in pseudo-polynomial time by a dynamic programming algorithm, even if preemption and set-up times are included. If job preemption is allowed without set-up times, TRP is solvable in polynomial time.

For BRP we adapted known maximum flow algorithms. We showed how the special structure of the maximum flow graph of BRP can be used to improve for BRP the general time complexity results of these algorithms. We also described a parametric maximum flow algorithm that solves BRP in the same time complexity as it solves the decision variant of BRP. Finally, we developed a linear time algorithm for the special case of random chained declustering. For TRP we designed two algorithms that construct a feasible solution out of the solution of the LP relaxation. For these algorithms we derived instance dependent performance bounds. We reported on the value of these bounds for several practical settings. Next to the LP based algorithms, we designed a linear time list scheduling algorithm.

The maximum flow algorithm solves BRP to optimality. With a probabilistic analysis we showed that with high probability a good load balance is obtained with this algorithm for BRP. Using the time-based approach we can improve on the results of the block-based approach, both in period length and in variation of the period length. The amount of improvement depends on a large number of parameters. We quantified with simulations the improvement for several settings. Furthermore, we illustrated with cases the effects of these improvements in disk efficiency on system performance parameters, such as response time and number of admissible clients. The results show that duplicate storage with a time-based algorithm enables exploitation of the multi-zone character of the disks by increasing the fraction of blocks that is read from the outer zones. Next to that, we showed that the fraction of time that disks are idle due to synchronization, turns out to be very small.

The time-based models and algorithms can be applied to a broad range of storage strategies and system settings. We showed that it works for random duplicate storage, partial duplication, and random striping. The models can also be adapted to hold for heterogeneous settings, such as heterogeneous disks or heterogeneous streams. A large advantage of random redundant storage strategies is that no assumptions have to be made about client behavior. By storing the blocks randomly, we make sure that in each period the blocks that have to be retrieved are a random choice out of the possible combinations. Consequently, the performance bounds on the period length hold without any assumptions regarding the requested blocks. Random redundant storage leads to very efficient disk usage, mainly at the cost of storage overhead. This means that if disk bandwidth is a scarce resource compared to disk storage capacity, random redundant storage is the preferred strategy for video-on-demand systems.

Bibliography

AERTS, J., J. KORST, AND S. EGNER [2000], Random duplicate storage strategies for load balancing in multimedia servers, *Information Processing Letters* **76**, 51–59.

AERTS, J., J. KORST, F. SPIEKSMA, W. VERHAEGH, AND G. WOEGINGER [2002], Load balancing in disk arrays: Complexity of retrieval problems, accepted for publication in *IEEE Transactions on Computers*.

AERTS, J., J. KORST, AND W. VERHAEGH [2001], Load balancing for redundant storage strategies: Multiprocessor scheduling with machine eligibility, *Journal of Scheduling* **4**, 245–257.

AERTS, J., J. KORST, AND W. VERHAEGH [2002], Improving disk efficiency in video servers by random redundant storage, *Proceedings Conference on Internet and Multimedia Systems and Applications (IMSA'02)*, 354–359.

AHUJA, R.K., T.L. MAGNANTI, AND J.B. ORLIN [1989], *Network flows*, Handbooks in Operations Research and Management Science 1, Optimization, Chapter IV, 211–370. Elsevier Science Publishers.

ALEMANY, J., AND J.S. THATHACHAR [1997], *Random striping for news on demand servers*, Technical Report TR-97-02-02, University of Washington.

AZAR, Y., A.Z. BRODER, A.R. KARLIN, AND E. UPFAL [1999], Balanced allocations, *SIAM Journal on Computing* **29**, 180–200.

BERENBRINK, P., A. BRINKMANN, AND C. SCHEIDELER [1999], Design of the PRESTO multimedia storage network, *Proceedings International Workshop on Communication and Data Management in Large Networks (CDM-Large'99)*.

BERENBRINK, P., A. CZUMAJ, A. STEGER, AND B. VÖCKING [2000], Balanced allocations: The heavily loaded case, *Proceedings Symposium on Theory of Computing (STOC'00)*, 745–754.

BERENBRINK, P., R. LÜLING, AND V. ROTTMANN [1996], A comparison of data layout schemes for multimedia servers, *Proceedings European Conference on Multimedia Applications, Services, and Techniques (ECMAST'96)*, 345–364.

BERENBRINK, P., M. A. RIEDEL, AND C. SCHEIDELER [1999], Simple competitive request scheduling, *Proceedings ACM Symposium on Parallel Algo-*

rithms and Architectures (SPAA'99), 33–42.

BERSON, S., S. GHANDEHARIZADEH, R.R. MUNTZ, AND X. JU [1994], Staggered striping in multimedia information systems, *Proceedings ACM SIGMOD Conference on Management of Data*, 79–90.

BERSON, S., R.R. MUNTZ, AND W.R. WONG [1996], Randomized data allocation for real-time disk I/O, *Proceedings IEEE COMPCON*, 286–290.

CHANG, E., AND H. GARCIA-MOLINA [1997], Effective memory use in a media server, *Proceedings Very Large Database Conference (VLDB'97)*, 496–505.

CHANG, E., AND A. ZAKHOR [1996], Cost analyses for VBR video servers, *IEEE Multimedia*, 56–71.

CHERVANAK, A.L., D.A. PATTERSON, AND R.H. KATZ [1995], Choosing the best storage system for video service, *Proceedings ACM multimedia*, 109–119.

CHUA, T.S., J. LI, B.C. OOI, AND K.-L. TAN [1996], Disk striping strategies for large video-on-demand servers, *Proceedings ACM Multimedia*, 297–306.

COFFMAN, E., L. KLIMKO, AND B. RYAN [1972], Analysis of scanning policies for reducing disk seek times, *SIAM Journal of Computing* **1**, 269–279.

DAN, A., D.M. DIAS, R. MUKHERJEE, D. SITARAM, AND R. TEWARI [1995], Buffering and caching in large-scale video servers, *Proceedings COMPCON*, 217–224.

DINIC, E. [1970], Algorithm for solution of a problem of a maximal flow in a network with power estimation, *Soviet Math. Doklady* **11**, 1277–1280.

FENG, W.C., AND J. REXFORD [1999], Performance evaluation of smoothing algorithms for transmitting prerecorded variable-bit-rate video, *IEEE Transactions on Multimedia* **1**, 302–313.

GALLO, G., M.D. GRIGORIADIS, AND R.E. TARJAN [1989], A fast parametric maximum flow algorithm and applications, *SIAM Journal on Computing* **18**, 30–55.

GAREY, M.R., AND D.S. JOHNSON [1979], *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Fransisco.

GEMMELL, J., H.M. VIN, D.D. KANDLUR, P.V. RANGAN, AND L.A. ROWE [1995], Multimedia storage servers: A tutorial, *IEEE Computer* **26**, 40–49.

GHANDEHARIZADEH, S., AND R.R. MUNTZ [1998], Design and implementation of scalable continuous media servers, *Parallel Computing* **24**, 91–122.

GOLDBERG, A.V. [1984], *Finding a maximum density subgraph*, Technical Report UCB CSD 84/171, University of California, Berkeley.

GOLDBERG, A.V., AND R.E. TARJAN [1988], A new approach to the maximum-flow problem, *Journal of the ACM* **35**, 921–940.

HSIAO, H., AND D.J. DEWITT [1990], Chained declustering: A new availability

strategy for multiprocessor database machines, *Proceedings International Conference on Data Engineering (ICDE'90)*, 456–465.

KARZANOV, A.V. [1974], Determining the maximal flow in a network with the method of preflows, *Soviet Math. Doklady* **15**, 434–437.

KORST, J. [1997], Random duplicated assignment: An alternative to striping in video servers, *Proceedings ACM Multimedia*, 219–226.

KORST, J., V. PRONK, AND P. COUMANS [1997], Disk scheduling for variable-rate data streams, *Proceedings European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'97)*, 119–132, LNCS 1309.

KORST, J., V. PRONK, P. COUMANS, G. VAN DOREN, AND E. AARTS [1998], Comparing disk scheduling algorithms for vbr data streams, *Computer Communications* **21**, 1328–1343.

LENSTRA, J.K., D.B. SHMOYS, AND E. TARDOS [1990], Approximation algorithms for scheduling unrelated parallel machines, *Mathematical Programming* **46**, 259–270.

LOW, C.P. [2002], An efficient retrieval selection algorithm for video servers with random duplicated assignment storage technique, *Information Processing Letters* **83**, 315–321.

LÜLING, R., AND F. CORTÉS GOMÉZ [1998], Communication scheduling in a distributed memory parallel interactive continuous media server system, *Proceedings Workshop on Architectural and OS Support for Multimedia Applications, in conjunction with ICPP'98*.

MARTELLO, S., AND P. TOTH [1990], *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley and Sons, New York.

MERCHANT, A., AND P.S. YU [1995], Analytic modeling and comparisons of striping strategies for replicated disk arrays, *IEEE Transactions on Computers* **44**, 419–433.

MICHIELS, W., J. KORST, AND J. AERTS [2002], On the guaranteed throughput of multi-zone disks, submitted to *IEEE Transactions on Computers*.

MOTWANI, R., AND P. RAGHAVAN [1995], *Randomized Algorithms*, Cambridge University Press.

MUNTZ, R.R., J.R. SANTOS, AND S. BERSON [1998], A parallel disk storage system for real-time multimedia applications, *International Journal of Intelligent Systems* **13**, 1137–1174.

NEMHAUSER, G.L., AND L.A. WOLSEY [1989], *Integer programming*, Handbooks in Operations Research and Management Science 1, Optimization, Chapter VI, 447–528. Elsevier Science Publishers.

NERJES, G., P. MUTH, AND G. WEIKUM [1997], Stochastic service guarantees for continuous data on multi-zone disks, *Proceedings ACM International*

Symposium on Principles of Database Systems (PODS'97).

OYANG, Y.-J. [1995], A tight upper bound of the lumped disk seek time for the SCAN disk scheduling policy, *Information Processing Letters* **54**, 355–358.

ÖZDEN, B., A. BILIRIS, R. RASTOGI, AND A. SILBERSCHATZ [1995], A disk-based storage architecture for movie on demand servers, *Information Systems* **20**, 465–482.

PAPADIMITRIOU, C.H., AND K. STEIGLITZ [1982], *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Inc., New Jersey.

PAPADOPOULI, M., AND L. GOLUBCHIK [1998], A scalable video-on-demand server for a dynamic heterogeneous environment, *Proceedings Workshop on Advances in Multimedia Information Systems, (MIS'98)*, Springer-Verlag, 4–17, LNCS 1508.

PATTERSON, D.A., G.A. GIBSON, AND R.H. KATZ [1988], A case for redundant arrays of inexpensive disks (RAID), *Proceedings ACM SIGMOD Conference on Management of Data*, 109–116.

PINEDO, M. [1995], *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Inc., New Jersey.

REHRMANN, R., B. MONIEN, R. LÜLING, AND R. DIEKMANN [1996], On the communication throughput of buffered multistage interconnection networks, *Proceedings ACM Symposium on Parallel Algorithms and Architectures (SPAA'96)*, 152–161.

RINNOOY KAN, A.H.G. [1987], Probabilistic analysis of algorithms, *Annals of Discrete Mathematics* **31**, 365–384.

RUEMMLER, C., AND J. WILKES [1994], An introduction to disk drive modeling, *IEEE Computer* **27**, 17–28.

SALEM, K., AND H. GARCIA-MOLINA [1986], Disk striping, *Proceedings International Conference on Data Engineering (ICDE'86)*, 336–342.

SANDERS, P. [2000], Asynchronous scheduling for redundant disk arrays, *Proceedings ACM Symposium on Parallel Algorithms and Architectures (SPAA'00)*, 98–108.

SANDERS, P. [2001], Reconciling simplicity and realism in parallel disk models, *Proceedings ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, 67–76.

SANDERS, P., S. EGNER, AND J. KORST [2000], Fast concurrent access to parallel disks, *Proceedings ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, 849–858.

SANTOS, J.R., R.R. MUNTZ, AND B. RIBEIRO-NETO [2000], Comparing random data allocation and data striping in multimedia servers, *Proceedings ACM Sigmetrics*, 44–55.

SCHOENMAKERS, L.A.M. [1995], *A new algorithm for the recognition of series*

parallel graphs, Technical report, CWI, Amsterdam.

SHENOY, P.J., P. GOYAL, S.S. RAO, AND H.M. VIN [1998], Symphony: An integrated multimedia file system, *Proceedings SPIE/ACM Conference on Multimedia Computing and Networking (MMCN'98)*, 124–138.

SHENOY, P.J., P. GOYAL, AND H.M. VIN [1995], Issues in multimedia server design, *ACM Computing Surveys* **27**, 636–639.

SHENOY, P.J., AND H.M. VIN [1999], Efficient striping techniques for variable bit rate continuous media file servers, *Performance Evaluation Journal* **38**, 175–199.

SHENOY, P.J., AND H.M. VIN [2000], Failure recovery algorithms for multimedia servers, *ACM Multimedia systems* **8**, 1–19.

TETZLAFF, W., AND R. FLYNN [1996], Block allocation in video servers for availability and throughput, *Proceedings SPIE/ACM Conference on Multimedia Computing and Networking (MMCN'96)*.

TOVEY, C.A. [1984], A simplified NP-complete satisfiability problem, *Discrete Applied Mathematics* **8**, 85–89.

VIN, H.M., S.S. RAO, AND P. GOYAL [1995], Optimizing the placement of multimedia objects on disk arrays, *Proceedings International Conference on Multimedia Computing and Systems (ICMCS'95)*, 158–165.

Author Index

Samenvatting

In een zogenaamd ‘video-on-demand’-systeem kunnen klanten op ieder moment een film naar keuze opstarten. De films liggen opgeslagen in een centrale ‘server’, en worden bij aanvraag over een extern netwerk naar de klant gestuurd. De server voorziet een groot aantal klanten tegelijk van hun eigen, continue stroom van video data. In een video server onderscheiden we drie delen: een verzameling harde schijven (disks) waarop de video data opgeslagen ligt, een geheugen van waaruit de data het externe netwerk ingestuurd wordt, en een intern netwerk dat de disks verbindt met het geheugen. Als een klant een film opstart krijgt hij een deel van de het geheugen als persoonlijke buffer toegewezen. Vanuit deze buffer wordt de video naar de klant gestuurd. De films worden opgesplitst in blokken van constante grootte en deze blokken worden op de disks opgeslagen.

We nemen aan dat de server in periodes werkt, en wel als volgt. Aan het begin van een periode wordt gekeken welke buffers ruimte hebben voor een volgend blok en de corresponderende blokken worden aangevraagd bij de disks. Ieder aangevraagd blok wordt toegewezen aan een disk, opgehaald, en verstuurd naar de corresponderende buffer. De volgende periode begint als alle disks klaar zijn met het ophalen van de aan hen toegekende blokken. In de server hebben we een algoritme nodig dat beschrijft hoe de data blokken worden opgeslagen op de disks en een algoritme dat de aangevraagde blokken toewijst aan de disks, waarbij de combinatie van beide algoritmen ervoor moet zorgen dat de disks efficiënt gebruikt worden.

In dit proefschrift analyseren we de werking van aselecte, redundante opslagstrategiëen. Van elk blok video data worden één of meer kopiëen opgeslagen op aselect gekozen disks. Voor de aangevraagde blokken die op meer dan één disk opgeslagen liggen, moet een keuze gemaakt worden welke disks te gebruiken voor ieder blok. Dit resulteert in het volgende zogenaamde ‘retrieval’ probleem dat in iedere periode opgelost dient te worden. Gegeven is een verzameling blokken en voor ieder blok is gegeven op welke disks het opgeslagen ligt. Ken nu de blokken toe aan de disks zodanig dat de periodelengte geminimaliseerd wordt. De verwachte periodelengte geeft aan hoe efficiënt de disks in de server gebruikt worden en is dus een maat voor de prestatie van een opslagstrategie met

bijbehorend retrieval-algoritme.

We beschouwen twee retrieval-problemen, die verschillen in de definitie van periodelengte. In het blok-gebaseerde retrieval-probleem (BRP) minimaliseren we het maximaal aantal blokken dat aan een van de disks is toegewezen, en in het tijdgebaseerde retrieval-probleem (TRP) minimaliseren we de daadwerkelijke eindtijd van de disk die het laatst klaar is. TRP is gebaseerd op een gedetailleerder model dan BRP. In TRP nemen we de daadwerkelijke leestijden van de blokken mee in de beslissing van toewijzing en staan we toe dat een blok gedeeltelijk van meer dan één disk opgehaald wordt. Het voordeel van het meenemen van de leestijden in het model is dat we bij het toekennen van de aangevraagde blokken aan de disks gebruik kunnen maken van de eigenschap dat magnetische schijven een hogere leessnelheid realiseren bij het lezen aan de buitenkant van de disk dan aan de binnenkant. De vrijheid om een blok in delen van meerdere disks te lezen heeft als voordeel dat het beter mogelijk is de hoeveelheid werk gelijkmatig te verdelen over de disks. Een nadeel is dat het totaal aantal verplaatsingen van de leeskoppen van de disks toeneemt.

We analyseren beide retrieval-problemen met technieken uit de combinatorische optimalisering. We laten zien dat de retrieval-problemen een speciale klasse van multiprocessor planningsproblemen vormen. We modelleren BRP als een geheeltallig lineair programmeringsprobleem en laten zien dat we het probleem kunnen oplossen met behulp van een speciale ‘maximum flow’ graaf. Dit betekent dat BRP oplosbaar is in polynomiale tijd. We modelleren TRP als gemengd geheeltallig lineair programmeringsprobleem en bewijzen dat het probleem NP-lastig is in de sterke zin. We beschrijven twee benaderings-algoritmen voor TRP, gebaseerd op LP-relaxatie, en een heuristiek gebaseerd op ‘list scheduling’.

Met een probabilistische analyse van BRP laten we zien dat gerandomiseerde redundante opslagstrategiëen goed presteren, in de zin dat de kans op een . Met simulaties kwantificeren we de verbetering van TRP ten opzichte van BRP. De resultaten geven aan dat TRP het mogelijk maakt de disks efficiënter te gebruiken, met name door gebruik te maken van de eigenschap dat disks sneller kunnen lezen aan de buitenkant dan aan de binnenkant. Aan de hand van een aantal toepassingen laten we zien hoe de toegenomen disk-efficiëntie gebruikt kan worden om, bijvoorbeeld, het aantal klanten toe te laten nemen.

Dit proefschrift laat zien dat aselecte redundante opslagstrategiëen een goede keuze zijn voor video-opslag in video-on-demand-systemen. De modellen en algoritmen zijn toepasbaar op een groot scala aan toepassingen en leiden tot zeer efficiënt disk gebruik.

Curriculum Vitae

Joep Aerts was born on 26 May 1975 in Riel, The Netherlands. He studied technical mathematics at the Technische Universiteit Eindhoven. He graduated with honors in April 1998, on the subject of test time reduction algorithms for core-based ICs. The Master's project was carried out at the Philips Research Laboratories in Eindhoven under supervision of Emile Aarts, Cor Hurkens, Jan Karel Lenstra, and Erik Jan Marinissen.

In May 1998 Joep started as a Ph.D. student at the Technische Universiteit Eindhoven. The research, that resulted in this thesis, was performed at the Philips Research Laboratories in Eindhoven under supervision of Emile Aarts, Jan Korst, and Wim Verhaegh.

Meyrueis, Lozère, 26 juni 1977. Warm, bewolkt weer. Ik pak mijn spullen uit mijn auto en zet mijn fiets in elkaar. Vanaf terrasjes kijken toeristen en inwoners toe. Niet-wielrenners. De leegheid van die levens schokt me.

[Tim Krabbé, De Renner]

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, Univ. Leiden. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01