Block placement on multi-zone disks

Michiels, W.P.A.J.

Award date:
1999

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computing Science

Master's Thesis

Block Placement on Multi-Zone Disks

W.P.A.J. Michiels

Supervisor: prof.dr. E.H.L. Aarts
Advisor: dr. ir. J.H.M. Korst

(6/1999)

Abstract

The objective is to reduce the cost per user of a video on demand system in which the users consume at a variable bit rate and the data blocks are of constant size. In a video server the video data is often stored on one or more hard disks. To increase their storage capacity, hard disks are nowadays partitioned into zones. A side-effect of zoning is that the disk has variable transfer rates. We present algorithms for maximize the guaranteed throughput over a given period of time by defining a placement of the data blocks on the disk. Furthermore, we revise the triple buffering and dual sweep disk scheduling algorithms, such that they are based on this throughput instead of the minimum guaranteed throughput. The result is a considerable reduction in the cost per user, both by a reduction in the required buffer capacity for a given number of users and by an increase of the number of users that can be serviced with the disk. This result also holds in relation to track pairing, which also utilizes the different transfer rates of a multi-zone disk. Although we also discuss how our approach can be used in combination with striping, we mainly focus on the case that the video data is stored on a single disk.

key words: video server, multi-zone disk, multimedia, constant block size, variable bit rate, cost per user

Acknowledgements

This master's thesis completes my studies at the Department of Mathematics and Computing Science of the Eindhoven University of Technology. The project is performed within the group New Media Systems & Applications at the Philips Research Laboratories in Eindhoven. I would like to thank my supervisors Emile Aarts and Jan Korst for their useful discussions and excellent comments. I would also like to thank my parents for all the support they gave me. Finally, I thank my colleagues for making my stay pleasant.

Wil Michiels,
Eindhoven, June 1999.

Contents

Chapter 1

Introduction

In a video-on-demand system multiple users can watch movies in an interactive way. This means that a user can choose which movie he/she wants to watch and at what time. Possibly, functions like pause/continue, slow motion and fast forward are also supported.

Figure 1.1 gives a model of a video-on-demand system. The system consists of a video server, multiple terminals and a communication network. The video server stores a collection of movies and must be able to offer each user the movie of his/her choice, which is to be sent as a continuous digital data stream over the communication network. When a user wants to interact with the server, he/she sends a request. Performance criteria of the system are response time, cost per user and reliability.

The video server often stores the movies on one or more hard disks. Although we discuss shortly the case that more disks are used, we mainly focus on the case where a single disk is used. Both the cost per user and the response time depend on the throughput of the used hard disk. Because the amount of data that can be stored on a single track increases with its distance to the spindle, many hard disks nowadays consist of several zones. A zone is a group of contiguous tracks. The tracks of a zone have an equal amount of data, but the amount of data per track in different zones increases with its distance to the spindle. Because a disk has constant angular velocity, different zones have different transfer rates. Most studies do not exploit these differences in transfer rate for designing a video-on-demand system, but assume the minimum guaranteed transfer rate of the disk. By alternately retrieving data from zones with a higher and lower transfer rate, a higher throughput can be guaranteed over a period of time.

In this thesis we aim at reducing the cost per user by defining a placement of the movies on a hard disk that distributes the successive blocks of a movie across the different zones. Thereby, we assume that a user consumes at a variable bit rate and that the amount of data that is retrieved from the disk during a single disk access is of constant size.

In the next two sections we discuss the model of a video server and consider in more detail how a hard disk can be modeled. Some notation is introduced in Section 1.3. The throughput of the disk influences the design of the video-on-demand system by means of the used disk scheduling algorithm. Therefore, we discuss in Section 1.4 some known disk scheduling algorithms. In Section 1.5 we state the problem that is the focus of this thesis. This chapter is concluded with some related work and an outline of the thesis.

1.1 Model of a video server

Figure 1.2 gives a model of a video server. Video data is often stored on hard disks, due to their large storage capacity and the possibility of random access. Because a hard disk can only perform one disk access at a time and because a server has to offer continuous data streams to several users, a buffer is reserved for

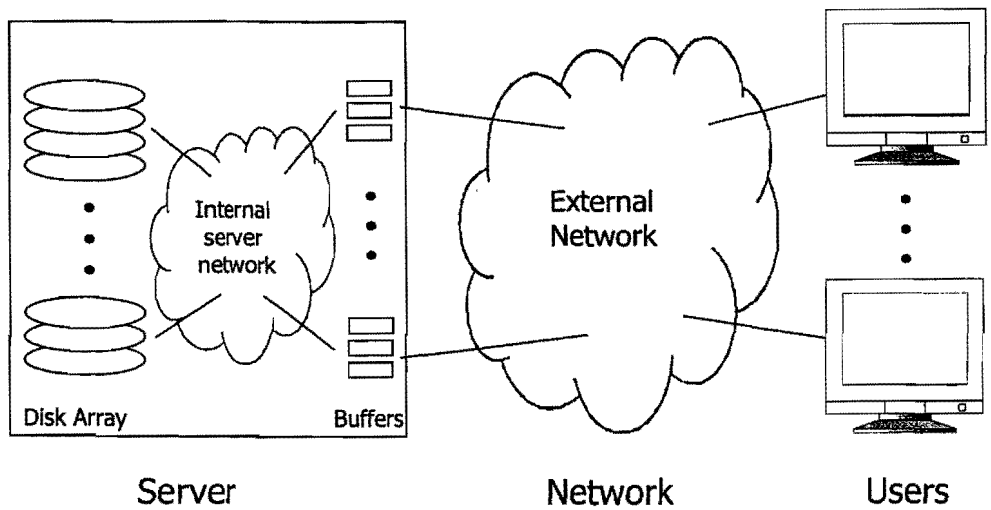

Figure 1.1. The video-on-demand system

each admitted user. A user can consume continuously from the corresponding buffer. Offering a user a continuous data stream consequently comes down to preventing that the corresponding buffer overflows or underflows. The task of scheduling the disk accesses is performed by a scheduler. In addition, the scheduler also has to handle user requests. A video-on-demand system is called safe if and only if it is guaranteed that neither buffer overflow nor buffer underflow occurs.

The filling of the buffers is done by periodically placing a data block in the buffer of each user. Several disk scheduling algorithms for the scheduler exist for determining on the basis of the degree of filling and the capacity of the buffers, which data blocks have to be fetched from the hard disk at a given moment. The choice of an appropriate algorithm predominately depends on whether the users consume at a variable or constant bit rate and whether data blocks of constant or variable size are being retrieved. In addition, the choice depends on which performance criterion should be emphasized, the response time or the cost per user. We define the response time as the time between the moment that a user request arrives at the server and the moment the user can start consuming the corresponding new data from the server. Note that this definition does not take into account the time required for transmitting a request from a user to the server and the time required for transmitting the data from the server to the user. The disk scheduling algorithm influences the cost per user by imposing a required buffer size for a given number of users and by giving a maximum number of admitted users that can be serviced with a given hardware configuration.

Two advantages of supporting variable bit rates are that variable-bit-rate-encoded data streams can be used, such as defined in the MPEG-2 standard [14], and that we get functions like slow motion and pause/continue without extra effort. On the other hand it has been shown that it also gives rise to larger buffer requirements [11]. An advantage of allowing variable block sizes is smaller buffer requirements [11]. However, it often results in a higher average response time and it poses some additional requirements on the storage of movies on disk [10].

In this thesis we consider the case that users consume at a variable bit rate that is bounded from above and data blocks are of constant size. We restrict ourselves to the homogeneous case, i.e., the case that all users have the same maximum consumption rate. Furthermore, we assume that the movies are stored on a single disk. In Chapter 7 we shortly discuss how our results can be generalized to the case that more disks are used. The model of a single hard disk is discussed in the next section.

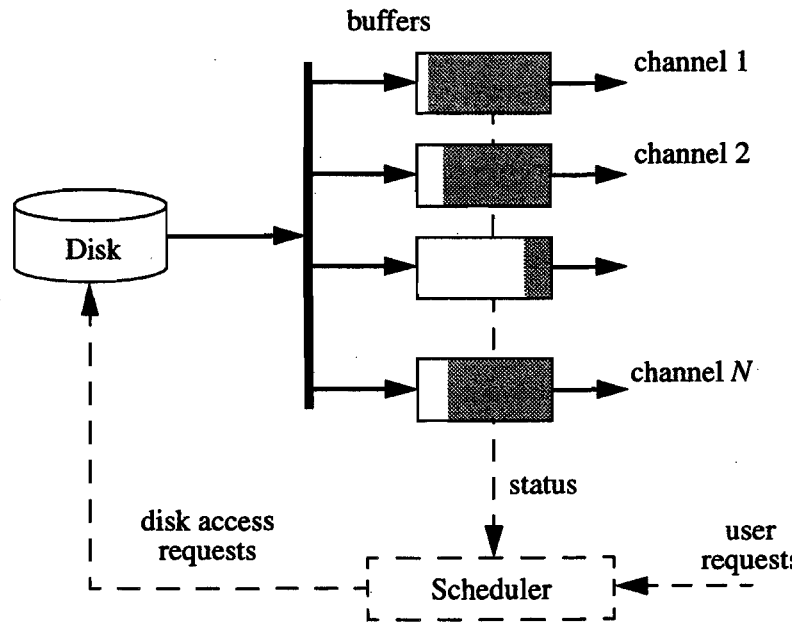

Figure 1.2. Model of the video server

1.2 Model of a hard disk

Figure 1.3 gives a schematic picture of a hard disk drive. A hard disk drive usually consists of several platters connected to the same spindle. Each platter consists of two surfaces on which data can be stored. Each surface has its own read/write head, where all heads are connected to the same actuator arm such that they have equal distance to the spindle. As a result, you can not move the heads separately. Furthermore, one can only read from one head at a time. The rate at which data is read is called transfer rate. The transfer rate is usually expressed in Mbit[1] per second.

Each disk surface is divided into concentric circles, called tracks. These tracks in turn are divided into an integer number of sectors. A sector is the smallest addressable unit of data for reading or writing. The size of a sector is often 512 Byte. The length of a track grows linearly with its radial distance to the spindle. Therefore, the outer tracks can contain more data, i.e., sectors per track, than the inner tracks. To exploit this, the set of all tracks of a hard disk is divided into contiguous groups, called zones. Each zone can consist of a different number of tracks. Within a zone, the number of sectors per track is kept constant, but between zones the number differs. Because a hard disk rotates at a constant angular velocity, this results in different transfer rates for different zones. For the Seagate Barracuda 9 drive, whose characteristics are given in Table 1.1, the highest transfer rate is 56% higher than the lowest transfer rate. Each surface has an equal number of tracks and corresponding tracks have an equal number of sectors. As a result, all heads are positioned at similar tracks. These tracks together are called a cylinder.

Each disk access (read or write) involves a sequence of successive sectors. Usually, the sectors are ordered cylinder by cylinder (see Figure 1.3). This means that successive sectors are as much as possible stored on the same cylinder instead of on the same platter. The reason is that the time required for moving to another track in the same cylinder, i.e., the time required for making another read/write head active, is usually smaller than the time required for moving the heads to the next track on the same surface. While moving to another track during a disk access, the first sector of the track might have rotated past the read/write head. Having arrived too late, the head should then have to stay in position until the first sector again arrives under the head. To avoid this loss of time, called rotational delay, the data on successive tracks is skewed.

[1] Here, we assume that 1 Mbit= 2^{20} bits and 1 MByte= 2^{20} Byte

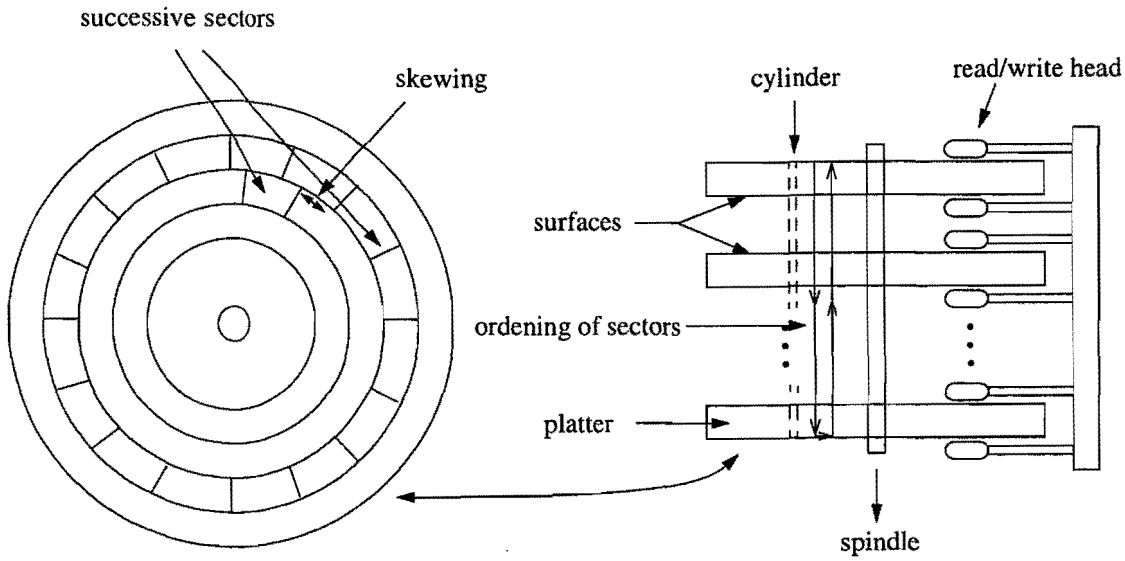

Figure 1.3. Schematic picture of a hard disk

This means that the first sector of each track is staggered in comparison to the first sector of the previous track to allow for the time the drive takes for the movement to the track.

The total time required for reading an amount of data generally consists of the following parts.

- The seek time. This is the time it takes the actuator to move to the right track, i.e., the track where the first sector to be read is positioned.
- Rotational delay. When a read/write head has arrived at the right track, it still has to wait until the first required sector is positioned under it. This time is at most the time required for one rotation.
- Track switch time. The time required for moving the read/write heads to the next track.
- Head switch time. The time required to make another read/write head active.
- Read time. The time that the disk is actually reading. This time is given by the amount of data that has to be read divided by the average transfer rate of the disk at the position where the data is stored.

We define the switch time as the time it takes to move a read/write head to the correct position. Hence, the switch time is the sum of the seek time and the rotational delay if data is stored on the surface of the active read/write head and it is the maximum of this sum and the head switch time, otherwise. Because the worst-case rotational delay is larger than the head switch time, the worst-case switch time equals the sum of the worst-case seek time and the worst-case rotational delay. Furthermore, we define the time required for a disk access as the transfer time. This time consists of read time and possibly a number of times the track switch time and a number of times the head switch time. The throughput of the disk is a measurement reflecting both the switch time and the transfer time. It represents the total amount of data that can be accessed during a unit of time. Because a hard disk rotates at a constant angular velocity, head switches and track switches appear at regular intervals during a disk access. Therefore, we can combine the transfer rate, the time required for the track switches and the time required for the head switches at a number of consecutive sectors of a disk into an adapted lower transfer rate, such that the actual transfer time is approximately given by the read time based on this adapted transfer rate. In the remainder of this thesis we assume this transfer rate, which thus implies that we can approximate the transfer time required for a disk access by the amount of data that has to be read divided by the transfer rate.

The seek time per access is maximally the time required for moving the head from the inner track to the outer track, or vice versa. To prevent that we have to take into account such a worst-case seek for each single access, disk accesses are usually made in batches. The disk then moves its head from the inside to

Seagate Barracuda 9 drive			
Drive Capacity	9.1 GByte	Minimum Transfer rate	59 Mbit/s
Platters	10	Maximum Transfer rate	92 Mbit/s
Surfaces	20	Average Transfer rate	79 Mbit/s
Rotational Speed	7200±0.5% r/min	Rotational Delay	8.38 ms
Sectors per Track	126-199	Track Switch Time	0.8 ms
Tracks per Surface	5273	Head Switch Time	1.2 ms
Number of Zones	7		

Table 1.1. Characteristics of the Seagate Barracuda 9 drive

the outside, or vice versa, and reads the blocks in the order in which they are encountered by the head. The execution of one such a batch of disk accesses is called a sweep.

Although a disk consists of several surfaces, we can model it as a single surface by considering a cylinder as a single track. The logical surface then has just as much tracks as a physical surface, but the capacity of a track increases. Let the number of surfaces of the disk be m. Because each track of a cylinder is divided into sectors in the same way, the capacity of a logical track is m times the capacity of a corresponding physical track. Hence, when we define the rotation speed as the original rotation speed divided by m, the transfer rate of a track of the logical surface equals the transfer rate of the corresponding track of a physical surface. As a result, we can assume that a disk consists of a single surface.

1.3 Notation

Before we present some known disk scheduling algorithms in Section 1.4, we introduce some notation that will be used throughout this thesis. The video server handles a maximum of n users, denoted by 1 through n. A user consumes at a variable bit rate anywhere between 0 and c_{max}. The constant size of a data block is denoted by B, which must be an integer number of sectors.

When B is given, we can indicate, say, n_B positions on the disk, where a data block can be stored. We number the positions in order of ascending transfer rate. Hence, for all $0 \leq i \leq j < n_B$ it holds that the time required for transferring a data block from position i is at least the time required for transferring a data block from position j. If B is not a divisor of the capacity of the disk, the positions do not cover the entire disk. We then leave a part of the disk with the lowest transfer rate unused. This is indicated by the shaded area in Figure 1.4. Because the transfer rate of a zone increases with its radial distance to the spindle, the numbering is as given in Figure 1.4. We assume that $n_B >> 1$ and we define F as the set of positions. Hence, $F = \{0, 1, \ldots, n_B - 1\}$.

As discussed in Section 1.2, the time it takes to execute a sweep consists of switch time and transfer time. We define $s_i(m)$ as the worst-case switch time required for fetching m data blocks in i sweeps. From the definition of switch time, it follows that $s_i(m)$ increases with m and with i. For $i = 1$, we usually omit the subscript i. It can be shown that $s_2(m) = s(\lceil \frac{m}{2} \rceil) + s(\lfloor \frac{m}{2} \rfloor)$, i.e., if the switching time required for m disk accesses in two sweeps is worst-case, then the disk accesses are equally divided over the two sweeps. When we generalize this, we get that $s_i(m)$ and $s(m)$ are related by

$$s_i(m) = \sum_{j=0}^{i-1} s\left(\left\lfloor \frac{m+j}{i} \right\rfloor\right). \tag{1.1}$$

We denote the time required for transferring the data block that is stored at position i, $0 \leq i < n_B$, by $t(i)$. By definition, the transfer time equals $\frac{B}{r(i)}$, where $r(i)$ is the average transfer rate at position i. From the numbering of the positions, it follows that t is an descending function. We define t_{max} as the maximum transfer time of any position and r_{min} as the minimum average transfer rate of any position. Hence, $t_{max} = t(0)$ and $r_{min} = r(0)$.

We consider the case that only one single movie is stored on the hard disk and that the movie consists of n_B

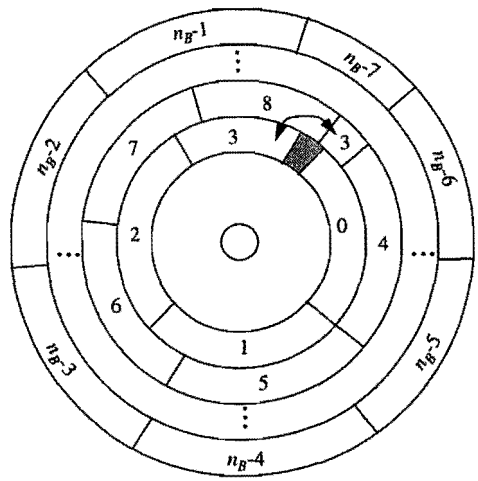

Figure 1.4. Numbering of the positions where data blocks can be stored

data blocks. This is not a restriction, as we show in Section 7.2. We number the data blocks of the movie from 0 through $n_B - 1$, where 0 is the first data block of the movie and $n_B - 1$ the last data block. We define L as the set of data blocks. Formally, $L = \{0, 1, \ldots, n_B - 1\}$. We introduce a variable i_u for each user u, $1 \leq u \leq n$, that indicates the first data block that will be fetched for u if no request arrives at the server until the start of the sweep in which data block i_u would be fetched. Consequently, when a data block has been fetched for user u, i_u is raised by one and when user u requests data block m instead of i_u, then the assignment $i_u := m$ is performed. When either the last data block of the movie has been fetched or a user is not admitted, then i_u is chosen equal to n_B, by definition. We define U as the set of users, for whom this is not the case. Hence, $U = \{u | i_u < n_B\}$. Consequently, U contains the users, for whom possibly a data block is to be fetched. Notice that both U and i_u, $1 \leq u \leq n$, are time dependent. Hence, we can only refer to them if a point in time is given.

We define the bijection $a : L \to F$, where $a(i) = j$ indicates that data block i is stored on position j. Hence, a describes how the movie is stored on the hard disk. We use t_a as a shorthand for $t \circ a$. Consequently, $t_a(i)$ gives the transfer time for fetching data block i. Using the notation, we get that the time required for a sweep in which the data blocks $y_1, y_2, \ldots, y_m$ are fetched, is at most $\sum_{j=1}^{m} t_a(y_j) + s(m)$. In the remainder we will frequently write the functions a and t_a as a sequence of numbers. Such a sequence has the straightforward meaning that $a(i)$ and $t_a(i)$, respectively, are at the ith position. These sequences are visualized by a chain of boxes where the ith box contains the ith value in the sequence. With a subsequence of a, we mean a number of successive data blocks with their corresponding positions. The length of a subsequence is called window size. Hence, the subsequence of a that starts at position i and has window size 2, refers to data block i and $i+1$ and thier corresponding positions $a(i)$ and $a(i+1)$.

Finally, we define the function $avg_f(A)$. This function gives the average of the function values of the elements of A with regard to the function f. Formally,

$$avg_f(A) = \frac{\sum_{x \in A} f(x)}{|A|}.$$

Using this definition the average time it takes to read a data block of the movie can be written as $avg_{t_a}(L)$ or, because a is a bijection, as $avg_t(F)$. This time will also be denoted by t_{avg}. Similar, we define r_{avg} as the average transfer time of the disk, i.e., the total used capacity of the disk divided by the transfer time required for transferring all data from disk.

We conclude this section with stating the convention that we define the minimum over an empty domain as ∞ and the maximum over an empty domain as $-\infty$.

1.4 Disk scheduling algorithms

The throughput of the disk influences the design of the video-on-demand system by means of the disk scheduling algorithm. In [11], three disk scheduling algorithms are presented for the case that users consume at a variable bit rate and data blocks are of constant size. Namely, triple buffering, dual sweep and m-EDFC. All three algorithms are based on the minimum transfer rate of the disk. For the first two algorithms we discuss in this thesis how they can be adjusted, such that another throughput than the minimum throughput is assumed. In this section we present these algorithms in their original form.

1.4.1 Triple Buffering

In the triple buffering algorithm (TB) the following strategy for filling the buffers of the admitted users is used.

Definition 1. (Filling Strategy TB). A data block is fetched for a user in a sweep if and only if the corresponding buffer has room for it at the start of the sweep. □

The size of a data block is chosen large enough to survive a worst-case sweep. In this case, a worst-case sweep is a sweep in which a data block is fetched for each user at a minimum transfer rate and in which the switching time is maximal. The time required for a worst-case sweep is consequently given by $n \cdot \frac{B}{r_{min}} + s(n)$. Hence, the block size satisfies

$$B \geq c_{max}(n \cdot \frac{B}{r_{min}} + s(n)). \tag{1.2}$$

The block size is minimal if equality holds in (1.2). As proved in [11], TB is safe if and only if the buffers of each user can store three data blocks. That this is a necessary condition can be inferred as follows. At the start of a given sweep w the buffer of a user can have room for $B - \varepsilon$ of data, with ε being a small positive value. As a result, no data block will be fetched for the user in sweep w. Hence, the next data block may not arrive before the end of sweep $w+1$. In the worst-case situation, two data blocks have been consumed during this time. Hence, a buffer must be large enough to store three data blocks.

Equation (1.2) can only be fulfilled if $c_{max} \cdot n < r_{min}$, i.e., the total rate at which data is consumed does not exceed the minimum transfer rate of the disk. As a result, the value of n is bounded by $\frac{r_{min}}{c_{max}}$.

A user may not start consuming before the end of the sweep in which the first data block is fetched. As a result, the worst-case response time of TB is the time required for two worst-case sweeps. This can be inferred as follows. If a user request arrives at the server, just after the start of a sweep, then the first data block for the user can not be fetched before the next sweep. Hence, it can take two worst-case sweeps before a user can start consuming.

1.4.2 Dual Sweep

In the dual sweep algorithm (DS) the following strategy for filling the buffers of the admitted users is used.

Definition 2. (Filling Strategy DS). A data block is fetched for a user in a sweep if and only if the corresponding buffer has room for it at the start of the sweep, unless a data block for this user has already been fetched in the previous sweep. □

The size of a data block is chosen such that it is large enough to survive two successive sweeps. Because in two successive sweeps at most one data block is fetched for each user, this means that

$$B \geq c_{max}(n \cdot \frac{B}{r_{min}} + s_2(n)). \tag{1.3}$$

has to hold. Again, the block size is minimal if equality holds and the value of n is bounded by $\frac{r_{min}}{c_{max}}$. The dual algorithm is safe if and only if the buffer of each user has room for two data blocks, as stated in [11]. Hence, dual sweep requires a data block less than triple buffering, although the data blocks will be somewhat larger than for triple buffering. The reason is that maximally one data block is consumed instead of two between the moment a user has room for a data block and the arrival of a data block, which, again, is maximally two sweeps.

As for triple buffering, a user can not start consuming before the end of the sweep in which the fist data block has been fetched. The worst-case response time for dual sweep is the time required for three worst-case sweeps. This can be inferred as follows. A request of a user can arrive just after the start of a sweep. Because in this sweep a data block can be fetched for the user and because a user may only fetch a data block once in two successive sweeps, it can take three sweeps before the user can start consuming. Note that in these sweeps, at most two data blocks are fetched for the same user.

1.5 Problem definition

As discussed in the previous section, in TB and DS, the blocks size is chosen, such that a data block is large enough to survive one and two worst-case sweeps, respectively. In a worst-case situation for each user a data block is fetched that is stored on the inner zone. Hence, when we assume a higher transfer rate we can service more users with the same block size and we can decrease the block size for the same number of users and correspondingly the buffer sizes. As a result, we can reduce the cost per user both due to an increase in the maximum number of admitted users and a reduction of the buffer requirements.

As discussed in Section 1.2 the minimum transfer rate can differ considerably from the average transfer rate. By placing the data blocks of a movie alternately on zones with a higher and lower transfer rate, sweeps that take a relatively long time can be alternated by sweeps that take a short time. Consequently, we can then guarantee a higher throughput whenever we consider a longer period of time than just one sweep, as for TB, or two sweeps, as for DS. This brings us to the following problem definition.

Problem definition 1. *Let a single movie be stored on a single multi-zone hard disk, let users consume at variable bit rate and let the data blocks be of constant size. Define a placement of the data blocks on the disk, such that a higher throughput of the disk can be guaranteed over a given period of time. In addition, revise the buffer requirements given by TB and DS, such that the relation between the block size and the maximum number of admitted users can be based on this higher throughput. The placement and the revision should aim at minimizing the cost per user.* □

1.6 Related work

In this section, we discuss related work on utilizing the different transfer rates of a multi-zone disk for increasing the throughput of the disk in a video-on-demand server.

By storing popular movies on the faster outer zones and less frequently watched movies on the inner zones, one obtains a better throughput statistically [4, 8, 15]. However, it does not increase the guaranteed throughput. Consequently, when safeness has to be guaranteed, the approach does not affect the design of the video-on-demand system but only improves the average performance of the video-on-demand system.

Several studies discuss region based data placement schemes to improve the guaranteed throughput of the disk [5, 7, 12]. In this approach, the disk is divided into regions and in each sweep, data blocks are read from only one region. As a result, the worst-case seek time is reduced and the guaranteed throughput over the period in which all regions are visited is increased. However, it results in relatively large worst-case response times, which can be inferred as follows. Consider the situation that a user requests a data block from the region from which the read/write head has just moved away. Then the user does not receive the requested data block before the head has visited all regions from which, in the worst-case situation, a data block is fetched for all other users. Furthermore, the approach assumes that a data block is fetched for the

consuming users in each period in which a region is visited. As a result, the approach is only suitable for normal playback, i.e., it is only suitable for systems without functions like fast forward and slow motion. Because the regions are visited in successive order, much buffer space is required to guarantee that no buffer under flow occurs in a period in which the inner regions are visited. In approach that we present, a period of several sweeps, in which only data blocks from the inner zones are fetched, is prevented.

In [16], the different transfer rates are utilized for the case that a user consumes at a variable bit rate. A constant period of time is allocated for retrieving each data block. As a result, the size of each data block is proportional to the data transfer rate and thus varies in different disk zones. The required buffer capacity is determined off-line and imposes predictive variable bit streams, i.e., the buffer requirements depend on both the playback time of the frames of a movie and their size. Hence, as for the region based approaches, the approach is only applicable for a system without functions like fast forward. Furthermore, it can only be used in a system with a static collection of movies.

Heltzer et al. [6] propose a scheme in which fixed-size logical tracks are constructed by comprising an equal number of same-numbered physical tracks from every zone. As a result, when the movies are stored in logical-track order, the read time is constant for each data block. Although the read time is based on the average transfer time of the disk, this is at cost of extra switch time. This overhead is partly alleviated by track pairing [1]. The amount of data on a track is approximately linear in its distance to the spindle. Consequently, when the innermost tracks are paired to the outermost tracks, each pair of tracks has approximately the same transfer rate. Consequently, by recording a movie alternately on a range of contiguous outer tracks and their inner counterparts, the guaranteed throughput of the disk is approximately based on the average transfer rate of the disk at the cost of some extra switching overhead. However, this switching overhead is smaller than for the approach presented by Heltzer et al. In Section 6.1, we discuss track pairing in more detail. Furthermore, we compare the results obtained by track pairing with our results.

1.7 Outline of this thesis

The organization of this thesis is as follows. In the next chapter we revise the buffer requirements given by TB and DS, such that the block size can be based on a higher throughput than the minimum throughput. In Chapter 3 we formulate the problem of minimizing the buffer requirements by means of appropriately placing the data blocks on the disk and we discuss the complexity of this problem. In addition, we give an approach to solve the problem, for which we present a solution strategy in Chapter 4. Chapter 5 discusses how requests from the users can be handled. In Chapter 6 we compare the cost per user resulting from our approach with the cost per user resulting from TB and DS. Furthermore, we compare our results with the results obtained by track pairing [1]. We discuss in Chapter 7 how our approach can be extended to a multi-disk model and that the assumption that only one single movie is stored on the disk that covers the entire disk is not restrictive. Finally, some concluding remarks are given in Chapter 8.

Chapter 2

Buffer requirements

In this chapter we derive how the buffer requirements of TB and DS can be adapted, whenever a higher throughput is assumed than the minimum throughput. After a user request arrives at the server, in both TB and DS the first requested data block is fetched as soon as it is allowed by the corresponding filling strategy. For TB, this means that the first requested data block is retrieved in the first sweep, say w, after the sweep in which the request arrives at the server. For DS, this is only the case if no data block has been fetched for the corresponding user in sweep $w-1$. Otherwise, the first requested data block is retrieved in sweep $w+1$. If we hold on to this strategy and a user requests in each sweep a data block that is stored on the inner zone of the hard disk, then each sweep is worst case in the case that filling strategy TB is used and each pair of sweeps is worst case in the case that filling strategy DS is used, regardless of how the movie is stored on the hard disk. As a result, the guaranteed throughput over a period of time equals the minimum throughput. Therefore, to assume another throughput than the minimum, we have to define additional conditions under which the first requested data block may be fetched. As mentioned in the problem definition, we focus on minimizing the cost per user. Hence, we do not want these conditions to affect the design of the video-on-demand system. Therefore, we assume that no user requests arrive at the server when designing the system. In Chapter 5 we discuss how requests can be handled, such that safeness remains guaranteed and the required buffer sizes do not increase.

In Section 2.1, we derive a sufficient buffer capacity for the case that filling strategy TB is used and a higher throughput is assumed than the minimum throughput. In Section 2.2, we do the same for filling strategy DS.

2.1 Revision triple buffering

As discussed in Section 1.4.1, in TB a data block is large enough to survive one worst-case sweep. A worst-case sweep in TB is a sweep in which a data block is fetched for each user at a minimum transfer rate and at maximum switch time. Whenever we assume a higher throughput than the minimum throughput, a data block is no longer large enough to survive a worst-case sweep. We therefore investigate the consequences of choosing an arbitrary B and n for the required buffer capacity of the users, where B is assumed to be too small to survive a worst-case sweep. Both in Section 2.1.1 and 2.1.2, we give a sufficient buffer capacity to guarantee safeness. Section 2.1.1 discusses the case that a data block is at least large enough to survive a sweep in which a data block is fetched for each user at maximum transfer rate. In Section 2.1.2 a more strict condition is imposed on the block size, namely that two data blocks are large enough to survive three successive sweeps.

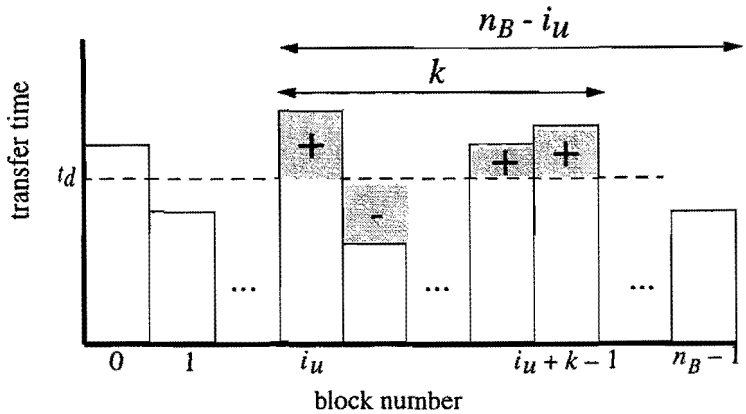

Figure 2.1. Interpretation of $\sigma_m(i)$.

2.1.1 Unconditional solution

For a given B and n, we define t_d, called the dimension time, as the transfer time that may be spent per user in a sweep, such that a data block is large enough to survive the sweep in the case that the switch time is maximal. Formally, t_d is defined such that $B = c_{\max}(n \cdot t_d + s(n))$. Hence, $t_d = \frac{B - c_{\max} \cdot s(n)}{c_{\max} \cdot n}$. By assumption, B is too small to survive a worst-case sweep. Consequently, $t_d < t_{\max}$. Figure 2.1 gives an example of the required transfer time for a given user u in k successive sweeps, whenever a data block is fetched for u in all k sweeps.

Let us first consider how safeness is guaranteed in the original TB algorithm before we consider the case that B is too small to survive a worst-case sweep. Safeness is guaranteed from a given sweep w onwards if and only if at the start of the sweep the buffers of the consuming users [1] contain at least one data block. This can be inferred as follows. If the buffer of a user contains less than one data block at the start of sweep w, then buffer underflow can occur when the sweep is worst-case because a data block may arrive just before the end of the sweep. Hence, a buffer has to contain at least one data block. Assume that the buffer of a user contains at least one data block at the start of sweep w. If a buffer contains at least one data block at the start of an arbitrary sweep, then, by definition, the buffer contains enough data to survive the sweep. Hence, it suffices to show that at the start of each sweep from sweep w onwards, the buffer of each user contains at least one data block. If a data block is fetched for the user in sweep w, then the buffer contains at least this data block at the start of sweep $w+1$. If, on the other hand, no data block is fetched for the user, then the corresponding buffer must have room for less than one data block, as follows from the definition of filling strategy TB. Consequently, the buffer contains at least two data blocks. During the sweep at most one data block is consumed. Hence, at the start of sweep $w+1$ the buffer contains again at least one data block. Hence, under the assumption that a buffer contains at least one data block at the start of sweep w, this also holds at the start of sweep $w+1$. By induction, we get that this means that at the start of each sweep w', $w' \geq w$, the buffer of each consuming user contains at least one data block, which has to be showed.

Now we consider how safeness is guaranteed in the case that a data block is too small to survive a worst-case sweep. The condition we have for the original TB algorithm is no longer sufficient, as can be verified as follows. Whenever a data block arrives at the end of a given sweep and the sweep is worst-case, then the buffer can underflow whenever it contains just one data block at the start of the sweep. We discuss the minimum number of data blocks a buffer has to contain at the start of a given arbitrary sweep w to prevent a buffer from underflowing. We thereby first consider the case that the required transfer time per sweep of just one user, say u, can deviate from t_d, i.e., only for user u the required transfer time for fetching the ith data block of the movie is given by $t_u(i)$. Hence, for the sake of this analysis, we assume $t_a(i) = t_d$ for all users except u. Furthermore, we assume that a data block is fetched for user u in each sweep from w

[1] In fact, the statement holds for consuming users for whom the last data block of the movie still has not been fetched.

onwards until the last data block of the movie, i.e. data block $n_B - 1$, has been fetched. Because $n_B - i_u$ data blocks have to be fetched for u from the start of sweep w, the last data block is fetched in sweep $w + n_B - i_u - 1$. We concentrate on preventing the buffers from underflowing during sweep w through $w + n_B - i_u - 1$, i.e., during the sweeps in which a data block is fetched for user u. Below, we will consider the general case where the transfer times for all the users vary.

As mentioned, a buffer has to contain at the start of a given sweep at least enough data to survive the sweep, because otherwise buffer underflow can occur whenever a data block is not placed in the buffer before the end of the sweep. The time required for a given sweep w', $w \leq w' < w + n_B - i_u$, can maximally be $(n-1) \cdot t_d + t_a(i_u) + s(n)$. Furthermore, a data block is large enough to survive $n \cdot t_d + s(n)$ time units. As a result, during sweep w' a user may have consumed one data block plus the number of data blocks that is consumed during $t_a(i_u) - t_d$ time units. This time is depicted in Figure 2.1 by the shaded block in a bar. Note that this time can both be positive and negative. When we divide $t_a(i_u) - t_d$ by the time that can minimally be survived with a single data block, we get the maximum number of data blocks that can be consumed more during sweep w' than a single data block. We define the time that can minimally be survived with a data block, which is $n \cdot t_d + s(n)$, by sb_u. Consequently, at the start of sweep w' a buffer has to contain at least

$$1 + \frac{t_a(i_u) - t_d}{sb_u} \tag{2.1}$$

data blocks.

We now investigate what the above implies for the number of data blocks the buffer of a user has to contain at the start of sweep w. Clearly, this has to be at least (2.1). However, we also have to guarantee that a user has the number of data blocks given by (2.1) in the corresponding buffer at the start of sweep $w+1$. Note that i_u is raised by one at the start of sweep $w+1$ in comparison with the start of sweep w because a data block has been fetched for u in sweep w. The fraction in (2.1) gives the number of data blocks a user may have consumed more during sweep w than the single data block that is placed in the buffer. Hence, to guarantee that (2.1) also holds at the start of sweep $w+1$, the buffer of a user has to contain at least

$$1 + \frac{t_a(i_u) - t_d}{sb_u} + \frac{t_a(i_u + 1) - t_d}{sb_u}$$

data blocks at the start of sweep w. Similarly, it can be showed that to guarantee that (2.1) holds at the start of sweep $w + k - 1$, $1 \leq k \leq n_B - i_u$, each buffer has to contain at least

$$1 + \frac{\sum_{j=0}^{k-1} (t_a(i_u + j) - t_d)}{sb_u} \tag{2.2}$$

data blocks at the start of sweep w. Consider Figure 2.1. The fraction in Expression (2.2) can be interpreted as the sum of the shaded blocks that are above the dotted line minus the sum of the blocks that are below it, divided by the time that can be survived with a single data block. The value k denotes the window size of the subsequence of a over which the value is calculated. When we maximize the expression over all values of k, $1 \leq k \leq n_B - i_u$, i.e., over all possible window sizes, we get the minimum number of data blocks a buffer has to contain, to prevent the buffer from underflowing in any sweep from w through $w + n_B - i_u - 1$. Let $\sigma_m(i)$ be defined by

$$\sigma_m(i) = \max_{m \leq k \leq n_B - i} \left(\sum_{j=0}^{k-1} (t_a(i + j) - t_d) \right). \tag{2.3}$$

Then, a buffer has to contain at least $1 + \sigma_1(i_u)/sb_u$ data blocks. It can be the case that no data block has been fetched for u in sweep $w-1$, because there was no room for it in the corresponding buffer. The time required for sweep $w-1$ is then $(n-1) \cdot t_d + s(n-1)$. Dividing by $n \cdot t_d + s(n)$ gives the maximum number of data blocks that is consumed during this sweep. To guarantee that the buffer of u contains at least $1 + \sigma_1(i_u)/sb_u$ data blocks at the start of sweep w, the buffer has to contain at least

$$1 + \frac{\sigma_1(i_u)}{sb_u} + \frac{(n-1) \cdot t_d + s(n-1)}{sb_u}$$

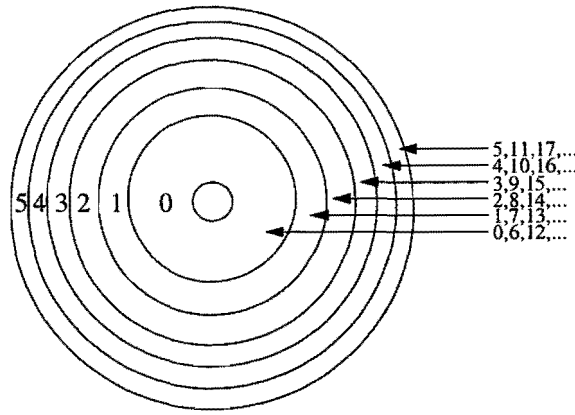

Figure 2.2. Placement of the data blocks

data blocks at the start of sweep $w-1$. To guarantee that when u has room for less than one data block in the buffer, the buffer still contains this number of data blocks, the buffer capacity has to be one plus this number. This and rewriting the fraction to $1+\frac{t_d+s(n)-s(n-1)}{sb_u}$ yields that the buffers have to be large enough to store

$$3+\frac{\sigma_1(i_u)}{sb_u}-\frac{t_d+s(n)-s(n-1)}{sb_u}. \tag{2.4}$$

data blocks. The interpretation of the fraction is the number of data blocks that is saved by fetching only $n-1$ data blocks instead of n. To guarantee that the buffer is large enough, regardless of the value of i_u, i.e., regardless of the data block that has to be fetched for user u in sweep w, we have to maximize (2.4) over all values of i_u. This yields

$$3+\max_{0\le i<n_B}\left(\frac{\sigma_1(i)}{sb_u}\right)-\frac{t_d+s(n)-s(n-1)}{sb_u}. \tag{2.5}$$

We give an example to clarify the above.

Example 1. We consider a hard disk that consists of six equal-sized zones, numbered from 0 through 5, and assume the worst-case switching time function of the Seagate Barracuda 9 drive. The transfer rate of the zones increases from 60 Mbit/s for the inner zone to 85 Mbit/s for the outer zone in steps of 5 Mbit/s. We assume that 12 users have to be serviced that have a maximum consumption rate of 4 Mbit/s. The worst-case switch time required for a sweep in which for each user a data block is fetched, i.e., $s(n)$, is 109.45 ms. Furthermore, $s(n-1)=101.17$.

We assume a block size of 171 kB. Furthermore, we assume that the data blocks are stored on the hard disk such that data block $i, 0\le i<n_B$ is stored on zone i mod 6 (see Figure 2.2). By definition, $t_d=\frac{B-c_{max}\cdot s(n)}{c_{max}\cdot n}$. Hence, $t_d=18.69$ ms. The transfer time required for fetching a data block can be calculated by dividing the block size by the transfer rate of the zone on which the data block is stored. Figure 2.3 depicts the transfer time of the data blocks.

Let i_u be 7 at the start of sweep w. In the figure $x=\sum_{j=0}^{2}(t_a(i)-t_d)$, where i mod $6=0$, i.e., data block i is stored on zone 0, and y is this expression for the case that i mod $6=3$, i.e., data block i is stored on zone 3. Hence, $x=(22.27-18.69)+(20.55-18.69)+(19.09-18.69)=5.83$. Similarly, $y=-5.85$. We first calculate how many data blocks have to be in the buffer at the start of sweep w, i.e., we calculate $1+\sigma_1(7)/sb_u$. Because $x<y$,

$$\max_{1\le k\le n_B-7}\sum_{j=0}^{k-1}(t_a(7+j)-t_d)=\sum_{j=0}^{1}(t_a(7+j)-t_d).$$

Hence, $\sigma_1(7)$ equals $(20.55-18.69)+(19.09-18.69)=2.25$ ms. Furthermore, $sb_u=12\cdot 18.69+109.45=333.73$ ms. As a result, $\sigma_1(7)/sb_u=0.00674$. Consequently, $1+\sigma_1(7)/sb_u=1.00674$.

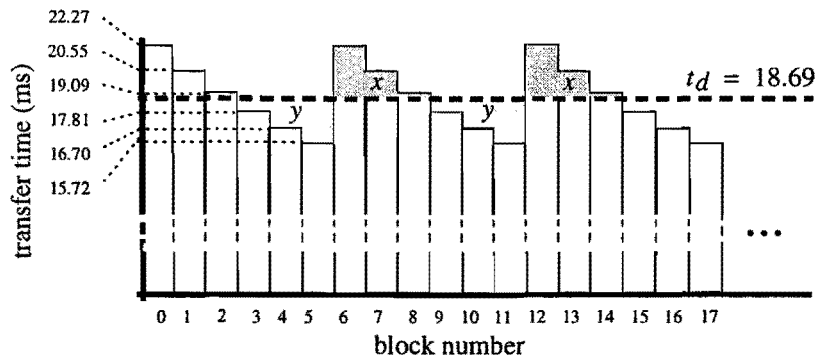

Figure 2.3. Transfer time of data blocks

Next, we determine the required buffer capacity, i.e., we determine the value of Expression (2.5). Again because $x < y$, $\max_{0 \le i < n_B} \sigma_1(i) = x = 5.83$. Hence, $\max_{0 \le i < n_B} \sigma_1(i)/sb_u = 5.83/333.73 = 0.01747$. Furthermore,

$$\frac{t_d + s(n) - s(n-1)}{sb_u} = \frac{18.69 + 109.45 - 101.17}{333.73} = 0.08081.$$

Consequently, (2.5) equals $3 + 0.00674 + 0.08081 = 3.088$. □

Assume that the transfer time of all users may deviate from t_d. Inspired by the discussion above, we require that at the start of a given sweep the buffer of each user contains at least

$$1 + \sum_{u=1}^{n} \frac{\sigma_1(i_u)}{sb_u}$$

data blocks. However, $\sigma_1(i_u)$ is only meaningful if $i_u \neq n_B$, i.e., if $u \in U$. Furthermore, buffer underflow can only occur in a buffer whenever the last data block of the movie still has not been placed in the buffer and the corresponding user is consuming. By definition, this means that we only have to define a minimum degree of filling for buffers of users from U. Hence, we require that at the start of the sweep the buffer of each user from U contains at least

$$1 + \sum_{u \in U} \frac{\sigma_1(i_u)}{sb_u} + (n - |U|) \cdot \frac{\tau}{sb_u}, \tag{2.6}$$

data blocks, where τ/sb_u is the contribution to the expression of the users that are not in U. We define τ as $t(n_B - 1) - t_d$. This will be useful in the discussion about how to handle user requests in Chapter 5. Note that since t is descending, $\tau \le \sigma_1(i)$ for all i, $0 \le i < n_B$.

Similar we can generalize the required buffer capacity given by (2.5) to

$$3 + n \cdot \max_{0 \le i < n_B - 1} \left(\frac{\sigma_1(i)}{sb_u} \right) - \frac{t_d + s(n) - s(n-1)}{sb_u}.$$

The following theorem states a sufficient buffer capacity to guarantee safeness in the case that filling strategy TB is used. The theorem is based on the obtained insight. For the proof we refer to Appendix A. Note that the buffer capacity is (mainly) determined by σ_2 instead of σ_1. Hence, the buffer capacity is (mainly) determined by the subsequences of a of length at least two instead of one.

Theorem 1. *If $t_d \ge t(n_B - 1)$, where $t_d = \frac{B - c_{max} \cdot s(n)}{c_{max} \cdot n}$, if no requests are sent by the users and if each user from U has initially at least the number of data blocks given by (2.6) in the corresponding buffer, then filling strategy TB is safe if there is room for at least*

$$3 + n \cdot \max_{0 \le i < n_B - 1} \left(\frac{\sigma_2(i)}{sb_u} \right) - \delta \tag{2.7}$$

data blocks, where σ_m is defined by (2.3) and

$$\delta = \max_{0 \le i < n_B - 1} \left(\frac{\sigma_2(i)}{sb_u} \right) - \max_{0 \le i < n_B} \left(\frac{\sigma_1(i)}{sb_u} \right) + \frac{t_d + s(n) - s(n-1)}{sb_u} \tag{2.8}$$

. □

We look at the meaning of the theorem for Example 1.

Example 1 revisited. As calculated in Example 1, $\max_{0 \le i < n_B}(\sigma_1(i)/sb_u) = 0.01747$ and $\frac{t_d + s(n) - s(n-1)}{sb_u} = 0.08081$. It can be verified that $\sigma_1(i) = \sigma_2(i)$. Hence, by (2.7), a sufficient buffer capacity is $3 + 12 \cdot 0.01747 + 0.08081 = 3.13$ data blocks, which equals 535.23 kB, because $B = 171$ kB.

From (1.2), it follows that the minimum block size for the original TB algorithm is 280.19 kB, which yields a minimum required buffer capacity of 840.57 kB. Hence, the required buffer capacity decreases by 36%. □

2.1.2 Conditional solution

In Subsection 2.1.1, we defined t_d such that one data block is large enough to survive a sweep in which the switch time is maximal and the transfer time required per user is t_d, i.e., $B = c_{\max}(n \cdot t_d + s(n))$. Alternatively, we now define t_d such that

$$B = c_{\max}(n \cdot t_d + \frac{1}{2} \cdot s_3(2n)) \tag{2.9}$$

or, equivalently, $t_d = \frac{B - \frac{1}{2} c_{\max} \cdot s_3(2n)}{c_{\max} \cdot n}$. Hence, the time that can minimally be survived with two data blocks is $2n \cdot t_d + s_3(2n)$. Consequently, two data blocks contain enough data to survive three sweeps if for a user at most two data blocks are fetched in three successive sweeps and the average transfer time required for transferring two successive data blocks for a user does not exceed t_d. In the next lemma we prove that the first condition is implied by the second one. We thereby consider an interval in which the hard disk is idle, because no data block has to be fetched, as an large number of sweeps. The condition on t_d can be formalized as

$$t_d \ge \frac{\max_{0 \le i < n_B - 1}(t_a(i) + t_a(i+1))}{2}. \tag{2.10}$$

The equation has the meaning that the size of a data block is no longer based on the worst-case required transfer time in a single sweep, as is the case in TB, but on the worst-case required transfer time in two sweeps.

Lemma 1. *If the users do not send requests and (2.10) holds, where $t_d = \frac{B - \frac{1}{2} c_{\max} \cdot s_3(2n)}{c_{\max} \cdot n}$, and if each user from U has initially room for at most one data block in the corresponding buffer, then in three successive sweeps at most two data blocks are fetched for the each user.*

Proof. We prove by induction to the number of executed sweeps, denoted by w, that in three successive sweeps never three data blocks are fetched for the same user. As basis, we take the cases that w is 1, 2 and 3. The induction hypothesis trivially holds for $w = 1$ and $w = 2$. Assume that $w = 3$ and assume that the induction hypothesis does not hold. This means that a user u, $1 \le u \le n$, exists, for whom a data block is fetched in sweep one, two and three. Hence, at the start of the first sweep u has to be an element of U, which, by assumption, implies that the buffer of u has room for at most one data block. The time required for the first two sweeps is maximal when in both sweeps a data block is fetched for all n users. This time does not exceed

$$n \cdot \max_{0 \le i < n_B - 1} (t_a(i) + t_a(i+1)) + 2s(n),$$

Because, by assumption, (2.10) holds, this is at most $2n \cdot t_d + 2s(n)$. Multiplying with the maximum consumption rate $c_{\max}$ yields

$$2c_{\max} \cdot (n \cdot t_d + s(n)), \tag{2.11}$$

which is the amount of data that is maximally consumed during the first two sweeps. From (1.1), it follows that $2s(n) = s_2(2n)$. Furthermore, it follows from the meaning of s_i that $s_2(2n) < s_3(2n)$. Hence, using (2.9), we get that in the first two sweeps, user u has consumed strictly less than two data blocks. Furthermore, two data blocks have been fetched during the first two sweeps. Consequently, the buffer contains strictly more data at the start of third sweep than at the start of first sweep, i.e., the buffer of u has room for strictly less than one data block at the start of third sweep. As a result, no data block is fetched for u in the third sweep, which yields the contradiction.

Consider the case $w = m + 1$, where $m \geq 3$. Let u be an arbitrary user, $1 \leq u \leq n$. Because the induction hypothesis holds for $w \leq m$, it suffices to prove that not in all three sweeps $m-1$, m and $m+1$ a data block is fetched for user u. Again, we prove it by contradiction. Hence, assume that in all three sweeps a data block is fetched. From the induction hypothesis and $m \geq 3$, it follows that in the sweeps $m-2$, $m-1$ and m at most two data blocks are fetched for the same user and that in sweep $m-2$ no data block is fetched for u because, by assumption, this is done in sweep $m-1$ and m. The first observation yields that the total time required for the sweeps $m-2$, $m-1$ and m does not exceed

$$n \cdot \max_{0 \leq i < n_B - 1} (t_a(i) + t_a(i+1)) + s_3(2n).$$

Because of the assumption that (2.10) holds, this is at most $2n \cdot t_d + s_3(2n)$. When we multiply this by $c_{\max}$ and divide it by B, which, by definition, is $c_{\max} \cdot (n \cdot t_d + \frac{1}{2}s_3(2n))$, we get that maximally two data blocks have been consumed during these sweeps. As mentioned, no data block is fetched for u in sweep $m-2$. Hence, u has room for strictly less than one data block in the buffer at the start of this sweep. Furthermore, a data block is fetched for u in both sweep $m-1$ and m. As a result, there is room for strictly less than one data block in the buffer of u at the end of sweep m, which means that in sweep $m+1$ no data block is fetched for user u. This gives the contradiction. This completes the proof of the induction step and correspondingly the proof of the lemma. □

In the next theorem we give the generalized version of Lemma 1. Because the proof is similar to the proof of the lemma, we omit its proof. In the theorem t_d is defined such that $B = c_{\max}(n \cdot t_d + \frac{1}{k}s_{k+1}(kn))$. Hence, k data blocks are large enough to survive $k+1$ sweeps if at most k data blocks are fetched for the same user in $k+1$ successive sweeps and the average transfer time required for a user for fetching k successive data blocks does not exceed t_d.

Theorem 2. *Let k be a positive integer. If the users do not send requests to the server,*

$$t_d \geq \frac{\max_{0 \leq i < n_B - 1} \sum_{j=0}^{k-1} t_a(i+j)}{k},$$

where $t_d = \frac{B - \frac{1}{k}c_{\max} \cdot s_{k+1}(kn)}{c_{\max} \cdot n}$, and if there is initially room for at most one data block in the buffer of each user, then in $k+1$ successive sweeps at most k data blocks are fetched for each user. □

Lemma 1 is the key to the following theorem, which gives a sufficient buffer capacity for the case that (2.10) holds. In the theorem, we denote the time that can minimally be survived with a single data block, i.e. $n \cdot t_d + \frac{1}{2}s_3(2n)$, by sb_c.

Theorem 3. *If the users do not send requests to the server, if Equation (2.10) holds and if each user from U has initially room for at most one data block, then filling strategy TB is safe if there is room for $3 - \delta_2$ data blocks, where*

$$\delta_2 = \frac{2t_d - t(0) + s_3(2n) - s_2(2n-1)}{sb_c}. \tag{2.12}$$

Proof. It follows from filling strategy TB, that no buffer overflow occurs. Furthermore, buffer underflow can only occur for users from U. Consider the moment that the buffer of an arbitrary user u, $u \in U$, turns into the state that it is able to store a data block. This means that there fits exactly one data block in the buffer. We prove that between this moment and the moment the buffer has (again) room for less than one data block, no buffer underflow occurs. Because initially each user from U has at most one free buffer place, this proves the theorem. It takes at most two sweeps before u receives the fist data block. This time is maximally

$$(n-1)\cdot \max_{0\le i<n_B-1}(t_a(i)+t_a(i+1)) + \max_{0\le i<n_B} t_a(i) + s_2(2n-1), \tag{2.13}$$

because these two sweeps contain only one data block for u. Hence, the buffer of u has to contain at least enough data to survive this time. Since t is descending and by (2.10), we get that this time is at most $2n\cdot t_d + s_3(2n) - \varepsilon$, where $\varepsilon = 2t_d - t(0) + s_3(2n) - s_2(2n-1)$. Dividing this by sb_c gives the number of data blocks that is maximally consumed during this time. By definition, this equals $2-\delta_2$, which is exactly the amount of data the buffer contains at the moment that we take under consideration. Hence, until the moment that user u receives his/her first data block, no buffer underflow occurs.

We still have to prove that no buffer underflow occurs between the moment that the first data block arrives in the buffer and the moment that u has less room in the buffer than one data block. Assume that the buffer of u receives the first data block in sweep w and that it has still room for more than one data block. This means that in sweep $w+1$ a data block is fetched for u, as well. From Lemma 1, it follows that in sweep $w+2$ no data block is fetched for user u. Consequently, between the arrival of the first data block and the moment the buffer has room for less than one data block, only one data block is fetched. This means that between these moments, the buffer has room for strictly less than two two data blocks. Hence, no buffer underflow occurs if the capacity of a buffer is at least two data blocks. This is the case, because if the capacity would be less than two data blocks, the buffer would have less than one free buffer place after receiving the first data block and this is in contradiction with the assumption that u has room for a data block after receiving the first one. □

2.2 Revision of dual sweep

The DS algorithm states that whenever a data block is large enough to survive two successive sweeps in which at most one data block is retrieved for each user, a buffer capacity of two data blocks suffices to guarantee safeness. By the used filling strategy, two successive sweeps are worst case when for each user one data block is fetched at a minimum transfer rate and the switch time is maximal. When we assume a higher throughput than the minimum throughput, a data block is no longer large enough to survive a worst-case situation of two successive sweeps. Hence, where we derived in the previous section a required buffer capacity for the case that a data block is too small to survive one worst-case sweep, we now have to derive a required buffer capacity for the case that a data block is too small to survive a worst-case situation of two successive sweeps. Therefore, let B and n be chosen arbitrary, where B is not large enough to survive a worst-case situation.

Again, we introduce a dimension time t_d. In Theorem 1, t_d has the meaning of the transfer time that may be spent per user, such that a data block is large enough to survive the sweep. We now define t_d as the transfer time that may be spent per user in two successive sweeps such that a data block is large enough to survive those sweeps. Although the new definition covers two sweeps instead of one, t_d is still related to the retrieval of a single data block, as follows from the definition of filling strategy DS. Formally, we define t_d such that $B = c_{\max}(n\cdot t_d + s_2(n))$ holds, i.e., $t_d = \frac{B - c_{\max}\cdot s_2(n)}{c_{\max}\cdot n}$. Furthermore, we define U_w, $w \ge 0$, as the set of users for whom a data block is fetched in sweep w. Note that, by the definition of filling strategy DS, $U_w \cap U_{w+1} = \emptyset$, for all $w \ge 0$.

Before we discuss a necessary buffer capacity to guarantee safeness for the case that a data block is not large enough to survive a worst-case situation of two successive sweeps, we consider how safeness is

guaranteed in the original DS algorithm. Therefore, we first investigate the number of data blocks that are maximally consumed during an arbitrary sweep w. The time that is maximally required for sweep w is given by $|U_w| \cdot t_{max} + s(|U_w|)$. When we divide this expression by the time that can maximally be survived with a single data block, i.e., $n \cdot t_{max} + s_2(n)$, we get the number of data blocks that is maximally consumed during sweep w. Hence, during sweep w maximally

$$\frac{|U_w| \cdot t_{max} + s(|U_w|)}{n \cdot t_{max} + s_2(n)} \tag{2.14}$$

data blocks are consumed. We can now state a necessary and sufficient condition to guarantee safeness for the original DS algorithm. Let the predicate $P(w)$ be defined as the property that at the start of sweep w each consuming user that is not an element of U_w has at least one data block in the buffer and each consuming user that is an element of U_w has at least the number of data blocks given by (2.14) in the buffer. Then safeness is guaranteed from a given sweep w onwards if and only if $P(w)$ holds. We first show that the necessary condition holds.

Assume that $u \notin U_w$ and that u has less than one data block in the buffer at the start of sweep w. Then the first data block may not arrive before the end of sweep $w+1$. In the worst-case situation one data block is consumed during sweep w and $w+1$. Consequently, buffer underflow occurs. Next, assume that $u \in U_w$ and u has less data blocks in the buffer than the number given by (2.14). We showed that (2.14) gives the number of data blocks that is maximally consumed during sweep w. Hence, buffer underflow can occur when the data block for u does not arrive before the end of the sweep.

Next, we prove the sufficient condition. Hence, we prove that if $P(w)$ holds, then safeness is guaranteed from sweep w onwards. Assume that $P(w)$ holds. If we show that $P(w')$ implies $P(w'+1)$ for all $w' \geq w$, then we get by the induction principle that $P(w')$ holds for all $w' \geq w$. Because $P(w')$ implies that each consuming user has at least the number of data blocks given by 2.14 in the corresponding buffer, no buffer underflow occurs in sweep w' if $P(w')$ holds. Consequently, it suffices to show that $P(w')$ implies $P(w'+1)$, $w' \geq w$.

Assume that $P(w')$ holds. We show that $P(w'+1)$ holds. We distinguish the cases that $u \notin U_w$ and that $u \in U_w$. Assume that $u \notin U_{w'}$, i.e., no data block is fetched for u in sweep w'. By $P(w')$, u has at least one data block in the buffer. Expression (2.14) with w replaced by w' gives the number of data blocks that is maximally consumed during sweep w'. Hence, at the start of sweep $w'+1$, u contains at least one data block minus (2.14). As mentioned, $U_{w'} \cap U_{w'+1} = \emptyset$. As a result, $n - |U_{w'}| \geq |U_{w'+1}|$ and $s_2(n) \geq s(|U_{w'}|) + s(|U_{w'+1}|)$. Consequently, one minus (2.14) is at least (2.14), where w is replaced by $w'+1$. Hence, if $u \in U_{w'+1}$, then $P(w'+1)$ holds. If, on the other hand, $u \notin U_{w'+1}$ then u has room for less than one data block, as follows from the definition of filling strategy DS and $u \notin U_{w'}$. Because the buffer has room to store two data blocks this implies that u has at least one data block in the buffer. Hence, $P(w'+1)$ holds.

Assume that $u \in U_{w'}$. Hence, a data block is fetched for u in sweep w'. Because $P(w')$ holds, u has enough data in the buffer to survive sweep w'. Furthermore, by the definition of filling strategy DS, the retrieved data block fits in the buffer. Hence, neither buffer underflow nor buffer overflow occurs in sweep w' and at the end of the sweep, u has at least one data block in the buffer, namely the retrieved data block. As a result $P(w'+1)$ holds.

We now consider how safeness is guaranteed in the case that a data block is too small to survive a worst-case situation of two successive sweeps. It is no longer the case that from a given sweep w safeness is guaranteed when $P(w)$ holds, as can be verified as follows. When a user u has exactly one data block in the buffer and $u \notin U_w$, then the first data block may not arrive before the end of sweep $w+1$. In the worst-case situation, more than one data block is consumed during sweep w and $w+1$. Consequently, buffer underflow can occur.

Similar as in the previous section, we first consider the case that for only one user, say u, the transfer time required for transferring a data block may deviate from t_d. Hence, when the ith data block is fetched for

user v, the corresponding transfer time is $t_a(i)$ if $v = u$ and t_d, otherwise. Furthermore, we assume that from a given sweep w until the sweep in which the last data block is fetched for user u, for each user a data block is fetched once in every two sweeps. Formally, we assume that $U_{w'} \cup U_{w'+1} = \{1,2,\ldots,n\}$, for each w', $w \leq w' \leq w_{last}$, where w_{last} is the sweep in which data block $n_B - 1$ is fetched for user u. There are $n_B - i_u$ data blocks to fetch for user u at the start of sweep w. As a result, $w_{last} = w + 2 \cdot (n_B - i_u) - 2$ if $u \in U_w$ and $w_{last} = w + 2 \cdot (n_B - i_u) - 1$, otherwise. By the assumption, we have that $U_w = U_{w+2} = U_{w+4} = \ldots$ and that $U_{w+1} = U_{w+3} = U_{w+5} = \ldots$.

We first derive how many data blocks are maximally consumed during two successive sweeps, say w' and $w' + 1$, $w \leq w' < w_{last}$. By assumption, a data block is fetched for each user in these sweeps. Hence, the time required for sweep w' and its successor can maximally be $(n-1) \cdot t_d + t_a(i_u) + s_2(n)$. Hence, the time required for sweep w' and $w' + 1$ is $t_a(i_u) - t_d$ more than the time that can be survived with a single data block. This time is depicted by the shaded blocks in Figure 2.1. Dividing this time by the time that can minimally be survived with a data block, gives the number of data blocks that is consumed more in sweep w' and $w' + 1$ than the number of data blocks that arrived in the buffer during the sweeps, i.e., one. Hence, the user has consumed

$$1 + \frac{t_a(i_u) - t_d}{sb_d} \tag{2.15}$$

data blocks during sweep w' and $w' + 1$, where sb_d is the time that can minimally be survived with a single data block, i.e. $sb_d = n \cdot t_d + s_2(n)$. Note that this expression has a similar meaning as Expression (2.1) in the previous section.

Using the above, we can derive the number of data blocks that have to be in the buffer of each user at the start of sweep w to prevent the buffer from underflowing during sweep w through w_{last}. We thereby assume that w_{last} is not about as large as w, i.e., there are still a couple of data blocks to fetch for user u. Let v be an arbitrary user. Assume that $v \notin U_w$. Then a data block may not arrive before the end of sweep $w + 1$. Consequently, the number of data blocks that are maximally consumed during sweep w and $w + 1$, which is given by (2.15), has to be in the buffer at the start of sweep w.

Because $v \notin U_w$ and because, by assumption, a data block is fetched for each user once in every two sweeps, $v \in U_{w+1}$ and $v \notin U_{w+2}$. Hence, to guarantee that no buffer underflow occurs in sweep $w + 2$ and $w + 3$, the buffer of v has to contain again at least the number of data blocks given by (2.15) at the start of sweep $w + 2$, where i_u is raised by one in comparison with the situation at the start of sweep w. As mentioned, the fraction in (2.15) gives the number of data blocks that are consumed more in two successive sweeps than the single data blocks that arrived in the buffer during the sweeps. Hence, to prevent the buffer of v from underflowing during sweep $w + 2$ and $w + 3$, the buffer has to contain at least (2.15) plus $\frac{t_a(i_u) - t_d}{sb_d}$ at the start of sweep w, which is given by

$$1 + \frac{t_a(i_u) - t_d}{sb_d} + \frac{t_a(i_u + 1) - t_d}{sb_d}.$$

Similarly, it holds that to guarantee that the buffer of v does not underflow in the sweeps in which data block $i_v + k - 1$ is fetched, i.e. in the sweeps $w + 2(k-1)$ and $w + 2(k-1) + 1$, the buffer of v has to contain at least

$$1 + \frac{\sum_{j=0}^{k-1} (t_a(i+j) - t_d)}{sb_d}$$

data blocks. This expression has a similar interpretation as Expression (2.2). When we maximize the expression over all possible values of the window size k, we get the minimum number of data blocks the buffer of v has to contain to prevent the buffer from underflowing in any sweep from w through w_{last}. By the definition of σ_m, this means that if $v \notin U_w$, then the buffer of v has to contain at least $\sigma_1(i_u)/sb_d$ data blocks.

Assume that $v \in U_w$, which means that a data block is fetched for v in sweep w. We, again, derive how many

data blocks the buffer at least has to contain to prevent the buffer from underflowing. The time required for sweep w is $|U_w \setminus \{u\} \cdot t_d + \chi_{(U_w)}(u)^2 \cdot t_a(i_u) + s(|U_w|)$. This can be written as $|U_w| \cdot t_d + s(|U_w|) + \chi_{(U_w)}(u) \cdot (t_a(i_u) - t_d)$. Dividing the expression by sb_d gives the number of data blocks that are maximally consumed during sweep w. Hence, maximally

$$\frac{|U_w| \cdot t_d + s(|U_w|)}{sb_d} + \chi_{(U_w)}(u) \cdot \frac{t_a(i_u) - t_d}{sb_d}. \tag{2.16}$$

data blocks are consumed during sweep w. Because $v \in U_w$, we have that $v \notin U_{w+1}$. Hence, as discussed the buffer has to contain at least $1 + \sigma(\tilde{i}_u)/sb_d$ data blocks at the start of sweep $w+1$ to prevent the buffer from underflowing from sweep $w+1$ through sweep w_{last}, where the tilde indicates that i_u is based on the situation before the start of sweep $w+1$. As a result, the number of data blocks the buffer of v contains at the start of sweep w minus the number of data blocks that is maximally consumed during sweep w, which is given by (2.16), plus the data block that is fetched during sweep w has to be at least $1 + \sigma_1(\tilde{i}_u)/sb_d$. Hence, the buffer of v has to contain at least the number of data blocks given by (2.16) plus $\sigma(\tilde{i}_u)$. However, when $\sigma(i_u)$ is negative, this number is smaller than the number given by (2.16). Hence, buffer underflow may occur during sweep w. Therefore we require that the buffer of v contains at least

$$\frac{|U_w| \cdot t_d + s(|U_w|)}{sb_d} + \chi_{(U_w)}(u) \cdot \frac{t_a(i_u) - t_d}{sb_d} + \frac{\max(\sigma_1(\tilde{i}_u), 0)}{sb_d} \tag{2.17}$$

data blocks in the case that $v \in U_w$. If $u \notin U_w$, then $\tilde{i}_u = i_u$ and $\chi_{(U_w)}(u) = 0$. Consequently, Expression (2.17) then equals

$$\frac{|U_w| \cdot t_d + s(|U_w|)}{sb_d} + \frac{\max(\sigma_1(i_u), 0)}{sb_d}. \tag{2.18}$$

Whenever $u \in U_w$, (2.17) is at most this expression. For the proof we refer to the proof of Theorem 4 in Appendix A from Expression (A.13). Hence, at the start of sweep w the buffer of an arbitrary user v has to contain at least $1 + \sigma_1(i_u)$ data blocks if $v \notin U_w$ and at least the number of data blocks given by (2.18) if $v \in U_w$. Note that Expression (2.18) does not have to be strict, as follows from its derivation.

We now discuss what the discussion above means for the buffer capacity. An arbitrary user v may not be an element of U_w both because the buffer has no room for a data block at the start of sweep w and because a data block has already been fetched for v in sweep $w-1$. Assume that the buffer has no room for a data block at the start of sweep w. As discussed, $v \notin U_w$ implies that the buffer has to contain at least $1 + \sigma(i_u)/sb_d$ data blocks. To guarantee this, the buffer capacity has to be at least one plus this expression. To guarantee that the buffer contains $2 + \sigma_1(i_u)/sb_d$ data blocks regardless of the value of i_u, we have to maximize the expression over all values of i_u. Hence, the buffer must be large enough to store at least

$$2 + \max_{0 \le i < n_B} \frac{\sigma_1(i)}{sb_d}$$

data blocks.

We return to the case that for each user the required transfer time may deviate from t_d. Because the users from U are the only users whose buffer possibly underflows, we only have consider the buffers of these users when defining a minimum degree of filling for the buffers. A straightforward generalization of the conditions above to guarantee that no buffer underflow occurs in any buffer from a given sweep w onwards, is to require that at the start of the sweep the buffer of an arbitrary user $u \in U$ contains at least

$$1 + \sum_{v \in U} \frac{\sigma_1(i_v)}{sb_d} \tag{2.19}$$

data blocks in the case that $u \notin U_w$ and at least

$$\frac{|U_w| \cdot t_d + s(|U_w|)}{sb_d} + \sum_{v \in U} \frac{\max(\sigma_1(i_v), 0)}{sb_d} \tag{2.20}$$

[2] $\chi_{(V)}(i) = \begin{cases} 1 & \text{if } i \in V \\ 0 & \text{if otherwise} \end{cases}$

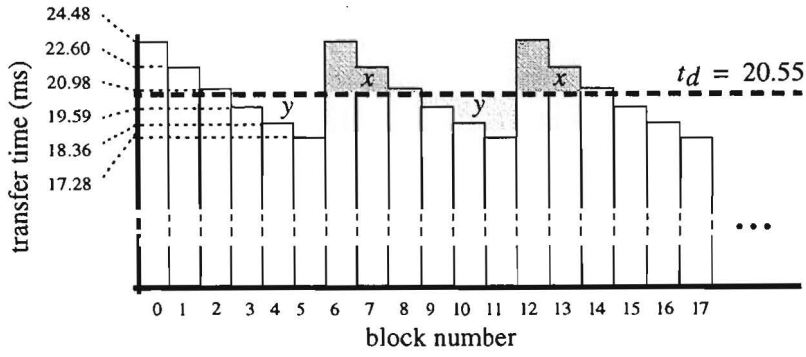

Figure 2.4. Example 2

data blocks otherwise. However, we require that an arbitrary user $u \notin U_w$ contains

$$1 + \sum_{v \in U} \frac{\max(\sigma_1(i_v), 0)}{sb_d} \tag{2.21}$$

data blocks, instead of the number given by (2.19). This has two reasons. Firstly, it is useful for the discussion about handling user requests. In addition, it has as consequence that (2.20) and (2.21) hold at the start of each sweep instead of only sweep w.

Similarly, we can generalize the buffer requirements to the case that the transfer time of each user may deviate from t_d. We get that a buffer must be large enough to store at least

$$2 + n \cdot \max_{0 \le i < n_B} \frac{\sigma_1(i)}{sb_d}$$

data blocks. The next theorem proves that safeness is indeed guaranteed with the discussed generalization. Unlike Theorems 1 and 3, the theorem also states that the given buffer capacity is necessary. For the proof we refer, again, to Appendix A

Theorem 4. *If $n \ge 2$ and $t(0) \ge t_d \ge t(n_B - 1)$, where $t_d = \frac{B - c_{max} \cdot s_2(n)}{c_{max} \cdot n}$, if the users do not send requests to the server and if each user from U has initially at least the number of data blocks given by (2.21) in the buffer, then filling strategy DS is safe if and only if there is room for at least*

$$2 + n \cdot \max_{0 \le i < n_B} \frac{\sigma_1(i)}{sb_d} \tag{2.22}$$

data blocks, where σ_m is defined by (2.3). □

We conclude this section with an example.

Example 2. We consider the same hard disk characteristics as in Example 1. Furthermore, we again assume that 12 users have to be serviced who have a consumption rate of 4 Mbit/s. The value of $s_2(n)$ is 120.50 ms. Consequently, by (1.3), the minimum block size in the original DS algorithm is 308.49 kB, which results in a buffer capacity of $2 \cdot 308.49 = 616.98$ kB.

We assume a block size of 188 kB. As a result, $t_d = 20.55$ ms. Figure 2.4 depicts the required transfer time for the data blocks in the case that the placement of the data blocks on the hard disk is similar as in Example 1. Both x and y have the same interpretation as in Example 1. It can be calculated that $x = 6.41$ ms and $y = 6.42$ ms. Hence, again $x < y$. As in Example 1, this yields that $\max_{0 \le i < n_B} \sigma_1(i) = x = 6.41$. By Theorem 4, safeness is guaranteed if and only if each buffer has room for at least the number of data blocks given by (2.22). Hence, a buffer must have room for at least

$$2 + \frac{x}{sb_d} = 2 + \frac{6.41}{12 \cdot 20.55 + 120.50} = 2.2095$$

data blocks. Because the block size is 188 kB, this means that the minimum buffer size is $2.2095 \cdot 188 = 415.39$ kB. As a result, the buffer requirements are decreased by 33% in comparison with DS. □

Chapter 3

Problem reformulations and their complexity

In Theorems 1, 3 and 4 we derived sufficient buffer capacities for the case that a higher throughput is assumed than the minimum throughput of the disk. In Theorems 1 and 4 the placement of the data blocks on the disk influences the required buffer capacity, and correspondingly the cost per user, by σ_2 and δ and σ_1, respectively. In Theorem 3, the placement determines whether condition (2.10) is satisfied.

In Section 3.1 we formulate the problem of defining a placement such that the buffer requirements given by Theorem 1 and Theorem 4 are minimized. Because this problem is NP-complete in the strong sense, as we will show, we relax the problem in Section 3.2. There, we also discuss the problem of finding an assignment such that condition (2.10) in Theorem 3 is satisfied.

3.1 Minimizing of the buffer capacity

In Theorem 1 the required buffer capacity increases linearly with $\max_{0 \leq i < n_B} \sigma_2(i)$ and, via δ, with the value of $\max_{0 \leq i < n_B} \sigma_1(i)$. Furthermore, the required buffer capacity is only influenced by the placement of the data blocks, i.e., the assignment a, by these expressions. Consequently, defining an assignment such that the required buffer capacity is minimized comes down to minimizing these expressions. We first show that it suffices to minimize the first expression, i.e., we show that whenever the first expression is minimized, the second is minimized as well.

To express $\sigma_1(i)$ in terms of $\sigma_2(i)$, $0 \leq i < n_B$, we split off $k = 1$ from the range of k in $\sigma_1(i)$. We get that

$$\sigma_1(i) = \begin{cases} \max(\sigma_2(i), t_a(i) - t_d) & \text{if} \quad i < n_B - 1 \\ t_a(i) - t_d & \text{if} \quad i = n_B - 1. \end{cases}$$

Consequently,

$$\max_{0 \leq i < n_B} \sigma_1(i) = \max\left(\max_{0 \leq i < n_B} \sigma_2(i), t_{\max} - t_d \right).$$

Because $t_{\max} - t_d$ is independent of assignment a, this implies that whenever $\max_{0 \leq i < n_B} \sigma_2(i)$ is minimized, $\max_{0 \leq i < n_B} \sigma_1(i)$ is minimized as well. As a result, minimizing the required buffer capacity given by Theorem 1 is equivalent to minimizing $\max_{0 \leq i < n_B} \sigma_2(i)$, which, by definition, equals

$$\max_{0 \leq i < n_B} \max_{2 \leq k \leq n_B - i} \sum_{j=0}^{k-1} (t_a(i+j) - t_d). \tag{3.1}$$

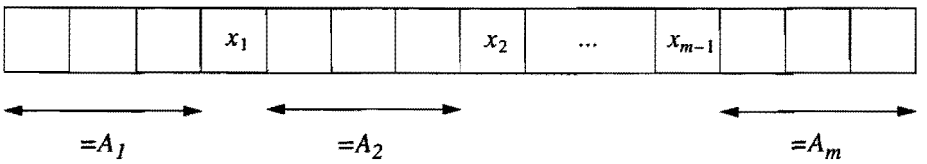

Figure 3.1. Assignment a

For an interpretation we refer to Figure 2.1. As discussed in Chapter 2, this expression can be interpreted as the maximum sum of the shaded blocks that are above the dotted line minus the sum of the shaded blocks that are below it, where the maximum is taken over all subsequences of a of length at least two.

In Theorem 4, the required buffer capacity increases linearly with the value of $\max_{0\leq i<n_B}\sigma_1(i)$ as well and, moreover, it is only influenced by assignment a by this expression. As a result, the buffer size given by Theorem 4 is also minimized once $\max_{0\leq i<n_B}\sigma_2(i)$ is minimized.

We define the problem of minimizing $\max_{0\leq i<n_B}\sigma_2(i)$ as a decision problem.

Problem definition 2. (Block Assignment Problem) *Given a set F of n_B physical addresses, a transfer time function $t : F \to \mathbb{N}^+$, a dimension time t_d and a bound $Z \in \mathbb{N}^+$. Is there a bijection $a : \{0,1,\ldots,n_B-1\} \to F$, such that*

$$\max_{0\leq i<n_B}\max_{2\leq k\leq n_B-i}\sum_{j=0}^{k-1}(t(a(i+j))-t_d)\leq Z \tag{3.2}$$

□

To prove that this problem is NP-complete in the strong sense, we make a reduction from 3-partition, which is proved to be NP-complete in the strong sense [3]. The problem is defined as follows.

Problem definition 3. (3-Partition) *Given a set A of $3m$ elements, a bound $Z \in \mathbb{N}^+$ and a size $s(a) \in \mathbb{N}^+$ for each $a \in A$, such that $\frac{Z}{4} < s(a) < \frac{Z}{2}$ and such that $\sum_{a\in A} s(a) = m \cdot Z$. Can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$, such that, for $1 \leq i \leq m$, $\sum_{a\in A_i} s(a) = Z$.* □

Theorem 5. *The block assignment problem is NP-complete in the strong sense.*

Proof. The storage capacity required for the sequence of a is polynomial in the storage capacity required for the input. Furthermore, it can be verified in polynomial time if a is a bijection and if 3.2 holds. This implies [13] that the assignment problem is in $\mathcal{NP}$.

Let I be an arbitrary instance of 3-partition. We define the function τ that maps I to an instance of our problem. We define the set of physical addresses, i.e. $F_{\tau(I)}$, as $A_I \cup \{x_1, x_2, \ldots, x_{m-1}\}$, the transfer time of x_i, $1 \leq i < m$, as 1 and the transfer time of the elements from A_I as $5Z_I + s_I(a) + 1$. Furthermore, we define the dimension time as $4Z_I + 1$ and $Z_{\tau(I)}$ as $4Z_I$. We have to prove that τ is a pseudo-polynomial transformation [3]. The transformation clearly can be performed in a time polynomial in the input length. It is also straightforward that the length of $\tau(I)$ is of the same magnitude as the length of I and the maximum value occurring in $\tau(I)$ is of the same magnitude as the maximum value occurring in I. What remains is the proof that I is a yes-instance of 3-partition if and only if $\tau(I)$ is a yes-instance of the assignment problem.

Let I be a yes-instance of 3-partition. Hence, the set A_I can be partitioned into m disjoint sets, each with total size Z_I. We will prove that $\tau(I)$ is a yes-instance of our problem, i.e. there exists a bijection a such that 3.2 holds. We prove this by constructing a. The first four elements of the sequence of a are the three elements from A_1 followed by x_1. The next four elements are the elements from A_2 followed by x_2. This is repeated until x_{m-1} is placed in the sequence. Finally, the elements of A_m are appended to the sequence. The obtained sequence is depicted in Figure 3.1.

Assignment a satisfies 3.2, if for all subsequences of a it holds that

$$\sum_{i\in S}(t_{\tau(I)}(i)-(t_d)_{\tau(I)})\leq Z_{\tau(I)}, \tag{3.3}$$

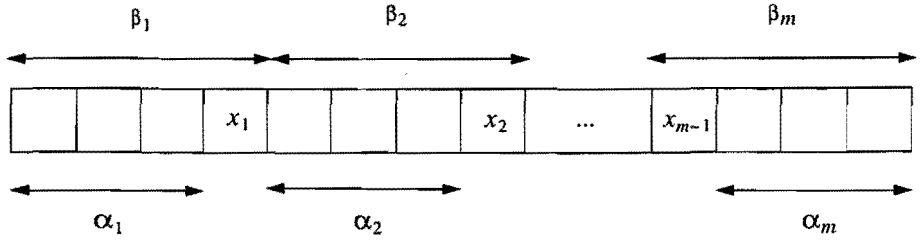

Figure 3.2. Definition α and β

where the set S contains the function values of a in this subsequence. Define the sequences α_i and β_i, $1 \leq i \leq m$, as depicted in Figure 3.2. Because the total size of an arbitrary set A_i, $1 \leq i \leq m$, eqauls Z_I, the value of the left hand side of (3.3) equals $4Z_I$ for sequence α_i and 0 for sequence β_i, as follows from the definition of $\tau(I)$. Furthermore, because s_I has a positive range, the value is less than $4Z_I$ for each subsequence of α_i and less than 0 for each subsequence of β_i that also contains x_i.

Consider an arbitrary subsequence. Clearly, there exists an i and a p, $1 \leq i \leq i+p \leq m$, such that the subsequence is given by

$$(\tilde{\beta}_i) ++ (\beta_{i+1}) ++ \cdots ++ (\beta_{i+p}) ++ (\tilde{\alpha}_{i+p+1}), \tag{3.4}$$

where $\tilde{\beta}_i$ and $\tilde{\alpha}_i$ are defined as a subsequence of β_i that contains x_i and a subsequence of α_i, respectively. Moreover, $sub_1 ++ sub_2$ defines the concatenation of sequence sub_2 and sequence sub_1. From the observations above, it follows that the left hand side of (3.3) for the sequence given by 3.4 is at most $4Z_I$. Hence, $\tau(I)$ is a yes-instance of the assignment problem.

Let $\tau(I)$ be a yes-instance of the assignment problem. This means that a bijection a exists such that (3.2) holds. We have to prove that I is a yes-instance of 3-partition. Assume that a contains a subsequence of four (or more) consecutive elements from A_I, starting at position i. We show that this leads to a contradiction. Because (3.2) holds, we have that

$$\sum_{j=0}^{3} (t_{\tau(I)}(a(i+j)) - (t_d)_{\tau(I)}) = 4Z_{\tau(I)}.$$

Using the definition of τ, the left-hand side can be written as $4Z_I + \sum_{j=0}^{3} s_I(a(i+j))$. Consequently, the equation is equivalent to

$$4Z_I + \sum_{j=0}^{3} s_I(a(i+j)) \leq 4Z_{\tau(I)}.$$

Because the range of s_I is $\mathbb{N}^+$, the left-hand side is strictly larger than $4Z_I$, which, by definition, equals $Z_{\tau(I)}$. This yields the contradiction. Hence, the sequence of a contains at most three consecutive elements from A_I. Because, by definition, $|A_I| = 3m$ and $|F_{\tau(I)} - A_I| = m - 1$, this can only be the case if a has m disjunct subsequences of length three, which consist of elements from A_I, separated by elements from $\{x_1, x_2, \ldots x_{m-1}\}$. Let V be the elements of such a subsequence. By the definition of τ,

$$\sum_{i \in V} (t_{\tau(I)}(a(i)) - (t_d)_{\tau(I)}) = 3Z_I + \sum_{i \in V} s_I(a(i)). \tag{3.5}$$

Because $\tau(I)$ is a yes-instance, the left-hand side is at most $4Z_I$. This and (3.5) yields that $\sum_{i \in V} s_I(a(i)) \leq Z_I$. Hence, a partitions the elements of A_I in m disjunct sets each containing three elements and with a total size of at most Z_I. Because the total size of A_I equals $m \cdot Z_I$, the total size of each set has to be exactly Z_I. Consequently, I is a yes-instance of 3-partition. □

3.2 Problem relaxation

Because the problem of minimizing (3.1) is NP-complete in the strong sense, we relax the problem. In the next subsection a relaxation is given that is based on reducing the number of subsequences of a by which the value of Expression (3.1) is determined.

3.2.1 Lowering the maximum windowsize

In TB and DS, the block size is based on the guaranteed throughput over a period in which maximally one data block is fetched per user. This period is one sweep in the case of TB and two sweeps in the case of DS. Let the block size be based on the guaranteed throughput over a period in which maximally k_{ub} data blocks are fetched per user. For filling strategy TB this means that a data block is large enough to survive the time that on average is required for one sweep, where the average is taken over k_{ub} successive sweeps. For filling strategy DS it means that a data block is large enough to survive the time that on average is required for two sweeps, where the average is taken over $2k_{ub}$ successive sweeps. In the first case the required buffer capacity is given by Theorem 1 and in the second case by Theorem 4. We investigate the consequences of such a block size for (3.1), which we have to minimize. We first write the condition on the block size in terms of a condition on the value of t_d.

Consider the case that filling strategy TB is used. The time required for k_{ub} successive sweeps is at most the time required for fetching k_{ub} successive data blocks for each user plus k_{ub} times the worst-case switch time required for a sweep, i.e., $k_{ub} \cdot s(n)$. Hence, the time required for k_{ub} successive sweeps is at most $n \cdot \max_{0 \le i \le n_B - k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j) + k_{ub} \cdot s(n)$. If there is a data block i_0 for which this expression is maximal and for which it holds that the scenario can occur that during k_{ub} successive sweeps for each user the data blocks i_0 through $i_0 + k_{ub} - 1$ are fetched, i.e., at the start of each of the k_{ub} sweeps, all buffers have room for at least one data block, then the time required for k_{ub} successive sweeps can be exactly the expression. Consequently, the condition that a data block is large enough to survive the time that on average is required for one of k_{ub} successive sweeps is implied by

$$B \ge c_{max} \cdot \frac{n \cdot \max_{0 \le i \le n_B - k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j) + k_{ub} \cdot s(n)}{k_{ub}}. \tag{3.6}$$

Because the buffer capacities are given by Theorem 1, t_d satisfies $B = c_{max}(n \cdot t_d + s(n))$. Consequently, (3.6) is equivalent to

$$t_d \ge \frac{\max_{0 \le i \le n_B - k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j)}{k_{ub}}. \tag{3.7}$$

Hence, t_d is at least the maximum transfer time that on average is required for transferring a data block, where the average is taken over k_{ub} successive data blocks.

Consider the case that filling strategy DS is used. The maximum time required for $2k_{ub}$ successive sweeps is at most the maximum sum of the transfer times required for transferring k_{ub} successive data blocks for each user plus the maximum switch time required for $2k_{ub}$ successive sweeps, which is $k_{ub} \cdot s_2(n)$. Hence, the time required for $2k_{ub}$ sweeps is maximal $n \cdot \max_{0 \le i \le n_B - k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j) + k_{ub} \cdot s_2(n)$. Dividing by k_{ub} gives the time that on average is required for two sweeps. Consequently, the condition on the block size is implied by

$$B \ge c_{max} \cdot \frac{n \cdot \max_{0 \le i \le n_B - k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j) + k_{ub} \cdot s_2(n)}{k_{ub}}. \tag{3.8}$$

Because the buffer sizes are given by Theorem 4, t_d satisfies $B = c_{max}(n \cdot t_d + s_2(n))$. Consequently, (3.8) can also be rewritten to (3.7).

From the above, it follows that if (3.7) holds, then a data block is large enough to survive the time that on average is required for one sweep when filling strategy TB is used, where the average is taken over k_{ub}

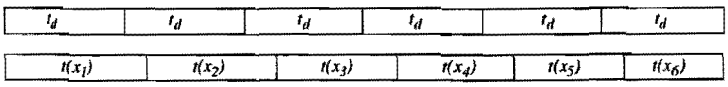

Figure 3.3. Example 1 revisited

successive sweeps, and it is large enough to survive the time that on average is required for two sweeps when filling strategy DS is used, where the average is taken over $2k_{\text{ub}}$ successive sweeps. Note that if $k_{\text{ub}} = 2$, then condition (3.7) is equivalent to (2.10) and if $k_{\text{ub}} = 1$, then (3.7) is equivalent to $t_d \geq t_{\max}$, which means that TB and DS are valid. We give an example to clarify the above.

Example 1 revisited. We show that (3.7) holds for $k_{\text{ub}} = 6$, i.e. we show that the transfer time that is worst-case required for transferring six successive data blocks does not exceed k_{ub} times t_d. Note that $x \leq y$ is a consequence of this property. Let V be an arbitrary set of six successive data blocks. By the definition of assignment a, in each of the six zones exactly one data block is stored that is an element of V. Let x_i, $0 \leq i \leq 5$, be the position in zone i on which an element of V is stored. As mentioned in Example 1, the transfer time required for transferring a data block from the zones 0 through 5 is 22.27, 20.55, 19.09, 17.81, 16.70 and 15.72 ms, respectively. In Figure 3.3, the transfer time corresponding to the positions x_1 through x_5 is depicted. It follows from the figure, that the sum of the six transfer times is less than $6t_d$. Because V is an arbitrary set of six successive data blocks, (3.7) holds for $k_{\text{ub}} = 6$. Consequently, a data block is at least large enough to survive the time that on average is required for one of six arbitrary successive sweeps, i.e., the block size is based on the guaranteed throughput over a period in which at most six data blocks are fetched for each user. □

It is easily verified that (3.7) is equivalent to

$$\max_{0 \leq i \leq n_B - k_{\text{ub}}} \sum_{j=0}^{k_{\text{ub}}-1} (t_a(i+j) - t_d) \leq 0, \tag{3.9}$$

which can be interpreted as follows. Consider Figure 2.1. Equation (3.9) is true if and only if for all k_{ub} successive bars, it holds that the sum of the shaded blocks that are above the dotted line is at most the sum of the shaded blocks that are below the line. In the next theorem we show that whenever (3.9) holds, the value of Expression (3.1) is determined by the subsequences of length at most k_{ub} instead of all subsequences. More precisely, we prove that if (3.9) holds, then the value of Expression (3.1) is between the value of this expression with the range of k replaced by 2 through $\min(k_{\text{ub}}, n_B - i)$ and the value of this expression with the range of k replaced by 1 through $\min(k_{\text{ub}}, n_B - i)$.

Theorem 6. *If for a given k_{ub}, $1 \leq k_{\text{ub}} \leq n_B$, (3.9) holds, then (3.1) is between*

$$\max_{0 \leq i < n_B} \max_{2 \leq k \leq \min(k_{\text{ub}}, n_B - i)} \sum_{j=0}^{k-1} (t_a(i+j) - t_d). \tag{3.10}$$

and

$$\max_{0 \leq i < n_B} \max_{1 \leq k \leq \min(k_{\text{ub}}, n_B - i)} \sum_{j=0}^{k-1} (t_a(i+j) - t_d). \tag{3.11}$$

Proof. Expression (3.1) is clearly at least Expression (3.10) because the range of k in (3.10) is a subset of the range of k in (3.1). We will now prove that (3.1) is at most (3.11).

Let i^* and k^* be the values, for which (3.1) is maximal. It suffices to show that there exist an i' and a k', where $0 \leq i' < n_B$ and $1 \leq k' \leq \min(k_{\text{ub}}, n_B - i')$, such that

$$\sum_{j=0}^{k^*-1} (t_a(i^* + j) - t_d) \leq \sum_{j=0}^{k'-1} (t_a(i' + j) - t_d). \tag{3.12}$$

If $k^* \leq k_{ub}$, this holds trivially. Assume that $k^* > k_{ub}$. Then there is a p and q, such that $k^* = p \cdot k_{ub} + q$, $p \geq 1$ and $q < k_{ub}$. Consequently, the left-hand side of (3.12) can be written as

$$\sum_{m=1}^{p} \sum_{j=0}^{k_{ub}-1} (t_a(i^* + m \cdot k_{ub} + j) - t_d) + \sum_{j=0}^{q-1} (t_a(i^* + p \cdot k_{ub} + j) - t_d). \tag{3.13}$$

By (3.9), $\sum_{j=0}^{k_{ub}-1} (t_a(i+j) - t_d) \leq 0$, $0 \leq i \leq n_B - k_{ub}$. Hence, (3.13) is at most $\sum_{j=0}^{q-1} (t_a(i^* + p \cdot k_{ub} + j) - t_d)$. Defining i' as $i^* + p \cdot k_{ub}$ and k' as q yields that (3.12) holds, which is to be proved. □

Clearly, (3.1) can be exactly (3.10) if (3.9) holds. In the next example we show that (3.1) can also be exactly (3.11) and can be strictly between (3.10) and (3.11).

Example 3. Let t_d be 5. Let t and a be such that the sequence t_a is given by 10 1 4 10. It can be verified that (3.9) holds for $k_{ub} = 3$, (3.1) is 5, (3.10) is 4 and (3.11) is 5. Consequently, (3.1) is exactly (3.11).

Let t_a be given by 10 1 4 7. Again, (3.9) holds for $k_{ub} = 3$. The values of (3.1), (3.10) and (3.11) are 2, 1 and 5, respectively. Hence, the value of (3.1) is strictly between (3.10) and (3.11). □

By Theorem 6, the number of subsequences of a by which the value of Expression (3.1) is determined decreases with the value of k_{ub}. Hence, it will generally be the case that the smaller the value of k_{ub} for which (3.9) holds, the smaller the value of Expression (3.1). As a result, we relax the problem of defining an assignment for which (3.1) is minimized, which we proved to be formally difficult, to the problem of defining an assignment for which (3.9) holds, where k_{ub} is minimized. Hence, we aim at minimizing of the length of the period in which the guaranteed throughput is at least the throughput on which the block size is based. We write this problem as a decision problem.

Problem definition 4. (Block Assignment Problem for Bounded Windowsize) *Given a set F of n_B physical addresses, a transfer time function $t : F \to \mathbb{N}^+$, a bound on the windowsize $1 \leq k_{ub} \leq n_B$ and a dimension time $t_d \in \mathbb{N}^+$. Is there a bijection $a : \{0, 1, \ldots, n_B - 1\} \to F$ and a windowsize k, $1 \leq k \leq k_{ub}$, such that*

$$\max_{0 \leq i \leq n_B - k} \sum_{j=0}^{k-1} (t(a(i+j)) - t_d) \leq 0 \tag{3.14}$$

□

We will use the abbreviation BAPBW to denote this problem. To prove that this problem is NP-complete, we will make a reduction from partition, for which we know that it is NP-complete [3].

Problem definition 5. (Partition) *Given a finite set A and a size $s : A \to \mathbb{N}^+$. Is there a subset $A' \subseteq A$ such that*

$$|A'| = \frac{|A|}{2} \text{ and } \sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)? \tag{3.15}$$

□

Theorem 7. *The block assignment problem for bounded windowsize is NP-complete.*

Proof. Similarly as in the proof of Theorem 5, it can be proved that BAPBW is in $\mathcal{NP}$. We define a function τ that maps an arbitrary instance of partition to an instance of BAPBW. Let I be an arbitrary instance of partition. If $|A_I| \bmod 2 = 1$, which implies that I is not a yes-instance because A_I cannot be partitioned into two equal sized sets, then we define $\tau(I)$ as an instance that is neither a yes-instance of $\tau(I)$. Let I be an instance, such that $|A_I| \bmod 2 = 0$. In the remainder of the proof α denotes

$$\left\lfloor \frac{\sum_{a \in A_I} s_I(a)}{2} \right\rfloor.$$

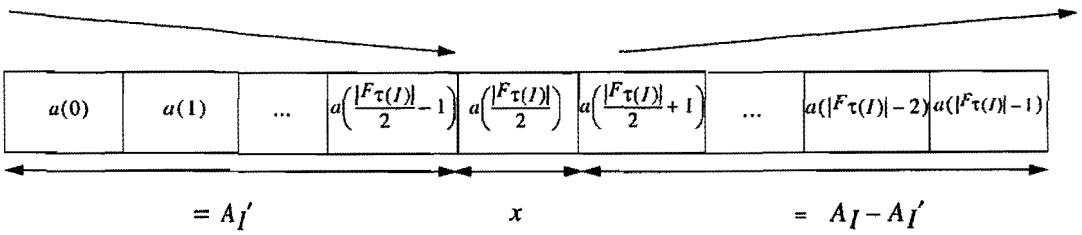

Figure 3.4. Assignment a

27	21	3	-52	9	12	30

Figure 3.5. Prove that $\tau(I^{ex})$ is a yes-instance of APBW

We define the set of physical addresses, i.e. $F_{\tau(I)}$, as $A_I \cup \{x\}$, $(k_{ub})_{\tau(I)}$ as $\frac{|A_I|}{2}+1$ and $(t_d)_{\tau(I)}$ as $((k_{ub})_{\tau(I)}-1)\cdot\alpha+1$. Furthermore, we define the transfer time function, such that $t_{\tau(I)}(v)$ is $((k_{ub})_{\tau(I)}-1)\cdot s_I(v)+(t_d)_{\tau(I)}$ if $v \neq x$ and $t_{\tau(I)}(v)=1$, otherwise.

We clarify the function τ by means of an example. Let I^{ex} be the instance of partition with $A_{I^{ex}} = \{a_0, a_1, \ldots, a_5\}$ and s such that the elements of $A_{I^{ex}}$ have a size of 1,3,4,7,9 and 10, respectively. The instance is a yes-instance of partition, because (3.15) holds for $A'_{I^{ex}} = \{a_0, a_3, a_4\}$. The total size of the set $A_{I^{ex}}$ is 34, which yields that $\alpha = 17$. By definition, we obtain that $F_{\tau(I^{ex})} = A_{I^{ex}} \cup \{x\}$, $(k_{ub})_{\tau(I^{ex})} = 4$, $(t_d)_{\tau(I^{ex})} = 3\cdot 17+1 = 52$ and $t_{\tau(I^{ex})}(v) = 3\cdot s(v)+52$ for all $v \in A_{I^{ex}}$.

We have to prove that τ is a polynomial-time transformation [13]. Clearly, $\tau(I)$ can be computed in a time polynomial in the length of I. Next, we prove that I is a yes-instance of partition if and only if $\tau(I)$ is a yes-instance of BAPBW. If $|A_I| \bmod 2 = 1$, then, by definition, neither I nor $\tau(I)$ are yes-instances. Therefore, we assume in the remainder that $|A_I| \bmod 2 = 0$.

Let I be a yes-instance of partition. Hence, there exists a set A'_I, such that (3.15) holds. We will prove that $\tau(I)$ is a yes-instance of BAPBW by defining a bijection $a : \{0,1,\ldots,|F_I|-1\} \rightarrow F_I$ and a $k \leq (k_{ub})_{\tau(I)}$, for which (3.14) holds. We define the first $\frac{|A_I|}{2}$ elements of the sequence of a as the elements of A'_I in order of a descending size. The element x is placed after these elements. The last $\frac{|A_I|}{2}$ elements are the elements of $A_I \setminus A'_I$ in order of ascending size (see Figure 3.4). Furthermore, $k = (k_{ub})_{\tau(I)} = \frac{|A_I|}{2}+1$. For our example, this means that the sequence of a is given by $a_4\ a_3\ a_0\ x\ a_1\ a_2\ a_5$ and that $k = 4$. Equation (3.14) holds if and only if for all subsequences of length k it holds that

$$\sum_{i \in S}(t_{\tau(I)}(i) - (t_d)_{\tau(I)}) \leq 0, \tag{3.16}$$

where S is the set containing the values of a in this subsequence. Figure 3.5 depicts the values of $t_{\tau(I^{ex})}(i) - (t_d)_{\tau(I^{ex})}$. It can be verified that (3.16) holds. Hence, $\tau(I^{ex})$ is a yes-instance of BAPBW. We now prove that this is also the case for $\tau(I)$.

We will first prove that the left side of Equation (3.16) is maximal for $S = A'_I \cup \{x\}$, i.e., for the subsequence of length k that starts at the first position. Consider a arbitrary subsequence of length k. Because the sequence of a has length $2\cdot k-1$, the subsequence has one of the first k elements of a as its first element. Consequently, it has the form given by Figure 3.6. This means that there exist four sets, V_1, V_2, W_1 and W_2, such that $V_1 \cup V_2 = A'_I$, $W_1 \cup W_2 = A_I - A'_I$ and $V_2 \cup \{x\} \cup W_1 = S'$, where S' is the set containing the values of the subsequence.

For proving that the left-hand side of (3.16) is maximal for $S = A'_I \cup \{x\} = V_1 \cup V_2 \cup \{x\}$, it suffices to prove

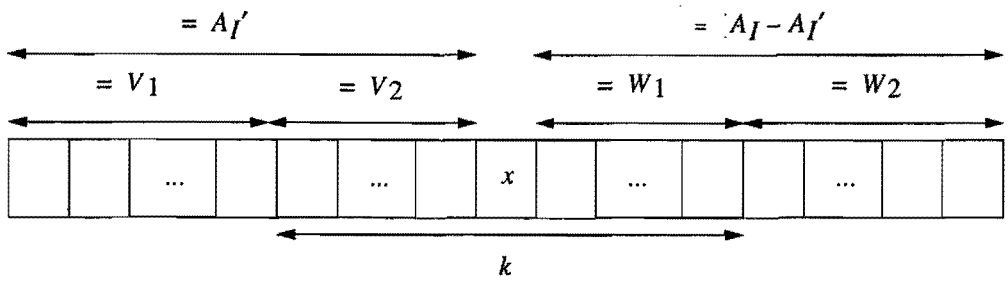

Figure 3.6. Definition of the sets V_1, V_2, W_1 and W_2

that

$$\sum_{i \in V_2 \cup \{x\} \cup W_1} (t_{\tau(I)}(i) - (t_d)_{\tau(I)}) \leq \sum_{i \in V_1 \cup \{x\} \cup V_2} (t_{\tau(I)}(i) - (t_d)_{\tau(I)}).$$

From the definitions of V_i and W_i, $i \in \{0,1\}$, and from the observation that $|A_I'| = k-1$, it follows that the cardinality of both V_1 and W_1 is $k - |V_2|$. By this and the definition of τ we get that the previous equation is equivalent to

$$\sum_{i \in W_1} s_I(i) \leq \sum_{i \in V_1} s_I(i).$$

Again because $|W_1| = |V_1|$, this is true if and only if $avg_{s_I}(W_1) \leq avg_{s_I}(V_1)$. From the way that a is constructed, it follows that $avg_{s_I}(A_I') \leq avg_{s_I}(V_1)$. Likewise, it holds that $avg_{s_I}(W_1) \leq avg_{s_I}(A_I - A_I')$. Because of (3.15) we also have that $avg_{s_I}(A_I') = avg_{s_I}(A_I - A_I')$. Consequently, $avg_{s_I}(W_1) \leq avg_{s_I}(V_1)$. As a result, the left-hand side of Equation (3.16) is maximal for $S = A_I' \cup \{x\}$.

Hence, proving that $\tau(I)$ is a yes-instance of BAPBW comes down to showing that (3.16) holds for $S = A_I' \cup \{x\}$, i.e. we have to prove that

$$\sum_{i \in A_I'} t_{\tau(I)}(i) + t_{\tau(I)}(x) - k \cdot (t_d)_{\tau(I)} \leq 0.$$

Using the definitions of τ, k and α, we can rewrite this to

$$\sum_{i \in A_I'} s_I(i) \leq \left\lfloor \frac{\sum_{a \in A_I} s_I(a)}{2} \right\rfloor.$$

The second equation in (3.15) implies that that the left-hand side equals the term inside the floor operator. Furthermore, the left-hand side is clearly integral. Consequently, the equation holds. Hence, $\tau(I)$ is a yes-instance of BAPBW.

Let $\tau(I)$ be a yes-instance of BAPBW. Let a be the assignment and k the windowsize for which (3.14) holds. We first show by contradiction that each subsequence of a of length k contains element x. Let S be the set containing the values of a subsequence of a of length k, such that $x \notin S$. Because, by definition, $A_I \neq \emptyset$, $(k_{ub})_{\tau(I)} > 1$. Furthermore, s_I has a strictly positive range. Hence, $t_{\tau(I)}(v) > (t_d)_{\tau(I)}$ for all $v \in A_I$. Consequently, (3.16) and correspondingly (3.14) do not hold, which gives the contradiction.

Because $|F_{\tau(I)}| = 2 \cdot (k_{ub})_{\tau(I)}$, a and k can only satisfy this property if $k = (k_{ub})_{\tau(I)}$ and element x is in the middle of the sequence of a, i.e.

$$a(\frac{|F_{\tau(I)}| - 1}{2}) = x.$$

Note that this is possible because $|A_I| \bmod 2 = 0$. We define C and D as the first $k-1$ and the last $k-1$

elements of the sequence of a, respectively. Formally,

$$C = \{a(0), a(1), \ldots, a(\frac{|F_{\tau(I)}|}{2} - 1)\}$$
$$D = \{a(\frac{|F_{\tau(I)}|}{2} + 1), a(\frac{|F_{\tau(I)}|}{2} + 2), \ldots, a(|F_{\tau(I)}|)\}.$$

Consequently, $F_{\tau(I)} = C \cup D \cup \{x\}$ and $A_I = C \cup D$. Equation (3.16) yields that

$$\sum_{i \in C} (t_{\tau(I)}(i) - (t_d)_{\tau(I)}) + t_{\tau(I)}(x) - (t_d)_{\tau(I)} \leq 0 \text{ and } \sum_{i \in D} (t_{\tau(I)}(i) - (t_d)_{\tau(I)}) + t_{\tau(I)}(x) - (t_d)_{\tau(I)} \leq 0.$$

Using the definition of τ and α gives that the equations are equivalent to

$$\sum_{i \in C} s_I(i) \leq \left\lfloor \frac{\sum_{a \in A_I} s_I(a)}{2} \right\rfloor \text{ and } \sum_{i \in D} s_I(i) \leq \left\lfloor \frac{\sum_{a \in A_I} s_I(a)}{2} \right\rfloor$$

Because $C \cup D = A_I$, the equations have to be equalities and the expression inside the floor operator has to be integral. Consequently, when we define A'_I as C, (3.15) holds. Hence, I is a yes-instance of partition. □

3.2.2 Solution approach

As proved, the relaxed problem discussed in the previous section is still formally difficult. In this subsection an approach is given to solve the relaxed problem, i.e., the problem of minimizing k_{ub} such that (3.9) holds. The main structure of the approach is as follows. For an increasing k_{ub} we check whether there exists an assignment a such that (3.9) holds. If this is the case, then the algorithm stops and outputs a. This clearly solves the problem. One step in the approach is still NP-complete, because the approach solves an NP-complete problem, as we proved, and the number of iterations is bounded by n_B, which is polynomial in the input length of the problem.

We discuss in more detail the problem of checking whether an assignment exists such that (3.9) holds for a given k_{ub}. We have already mentioned that (3.9) is equivalent to (3.7). Furthermore, rewriting (3.7) gives that there exists an assignment a such that (3.7) holds if and only if there exists an assignment a such that

$$\max_{0 \leq i \leq n_B - k_{\text{ub}}} \sum_{j=0}^{k_{\text{ub}}-1} t_a(i+j) \tag{3.17}$$

does not exceed $k_{\text{ub}} \cdot t_d$, i.e., such that the maximum sum of the transfer times of k_{ub} successive data blocks does not exceed $k_{\text{ub}} \cdot t_d$. Hence, the problem of checking whether there exists an assignment such that (3.9) holds is equivalent to the problem of checking whether there exist an assignment such that (3.17) does not exceed $k_{\text{ub}} \cdot t_d$. This problem is solved by minimizing (3.17) and checking whether the expression exceeds $k_{\text{ub}} \cdot t_d$. This problem is still NP-complete because it solves a NP-complete problem. To solve the problem of minimizing (3.17), we use a heuristic. Note that as a consequence of using a heuristic, an assignment that satisfies (3.9) for a given k_{ub} may not be found, while it does exist.

Whenever (3.9) holds for $k_{\text{ub}} = 1$, then $t_d \geq t_{\max}$ holds, i.e., if t_d is defined as in Theorem 1, then a data block is large enough to survive one worst-case sweep and if t_d is defined as in Theorem 4, then a data block is large enough to survive the time that is in the worst-case situation required for two successive sweeps. However, we assumed that this is not the case. Consequently, we can take two as starting value of k_{ub}. Algorithm 1 depicts the approach we use to solve the problem of minimizing k_{ub} such that (3.9) holds. In the ith iteration of the algorithm the problem of finding an assignment for which (3.7) holds for $k_{\text{ub}} = i + 1$ is solved. As mentioned in the previous section, condition (2.10) in Theorem 3 is equivalent to (3.7) if $k_{\text{ub}} = 2$. Hence, executing only the first iteration of the algorithm, solves the problem of finding an assignment such that (2.10) is satisfied.

The next theorem states an lowerbound on the value of (3.17) for the case that k_{ub} is a divisor of n_B.

Algorithm 1 Approach

repeat
  if ($k_{ub} = \perp$)
    $k_{ub} := 2$;
  else
    $k_{ub} := k_{ub} + 1$;
  construct an assignment a, such that $x := \max_{0 \le i \le n_B - k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j)$ is minimized;
until $x \le k_{ub} \cdot t_d$ or $k_{ub} = n_B$;

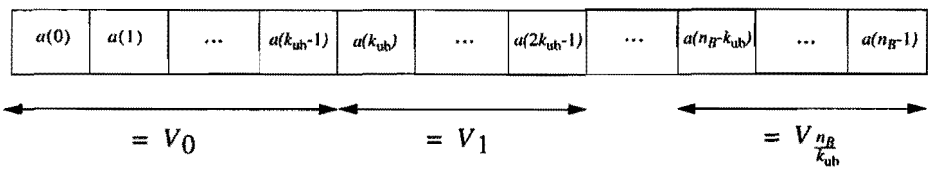

Figure 3.7. Definition of V_i

Because (3.17) is the lowerbound of $k_{ub} \cdot t_d$, this also gives a lowerbound on the value of t_d for which an assignment that satisfies (3.9) possibly exists.

Theorem 8. *For each windowsize k_{ub} that is a divisor of n_B and each assignment a,*

$$\max_{0 \le i \le n_B - k} \sum_{j=0}^{k_{ub}-1} t_a(i+j) \ge k_{ub} \cdot t_{avg}. \tag{3.18}$$

Proof. We prove the theorem by contradiction. Hence, let k_{ub} be a divisor of n_B and let a be an assignment, such that (3.18) does not hold. Define $V_0, V_1, \ldots, V_{\frac{n_B}{k_{ub}}-1}$ as depicted in Figure 3.7. Formally, $V_i = \{a(i \cdot k_{ub}), a(i \cdot k_{ub}+1), \ldots, a((i+1) \cdot k_{ub}-1)\}$, for all i, $0 \le i < \frac{n_B}{k_{ub}}$. Because (3.18) does not hold,

$$\sum_{j \in V_i} t(j) < k_{ub} \cdot t_{avg},$$

for each set V_i, $0 \le i < \frac{n_B}{k_{ub}}$. When we sum over all sets, we get that

$$\sum_{i=0}^{\frac{n_B}{k_{ub}}-1} \sum_{j \in V_i} t(j) < n_B \cdot t_{avg}.$$

Because a is a bijection and because $V_i \cap V_j = \emptyset$, for all $0 \le i < j < \frac{n_B}{k}$, the left-hand side is exactly the sum of the transfer times of all elements of F. The right-hand side can be interpreted similarly, as follows from the meaning of t_{avg}, which yields the contradiction. □

As a result of the theorem, there does not exist an assignment, such that (3.9) holds whenever $t_d < t_{avg}$ and k_{ub} is a divisor of n_B. The theorem does not have to be valid if n_B is not divisible by k_{ub}, which we will show by a counter example. Consider a hard disk with three places and with a transfer time function that assigns the value 2 to the inner two places and the value 1 to the outer place. Hence, its average transfer time is $1\frac{2}{3}$ units of time. We will show that there exists an assignment a for which (3.18) is true, for $k_{ub} = 2$. Define $a(0)$, $a(1)$ and $a(2)$ as 0, 2 and 1, respectively. The left side of (3.18) can now be written as $\max(t(0)+t(2), t(2)+t(1))$, which is 3. However, $k_{ub} \cdot t_{avg} = 3\frac{1}{3}$. Hence, (3.18) does not hold.

For all $k_{ub} \le n_B$, $n_B - (n_B \bmod k_{ub})$ is divisable by k_{ub}. Consequently, by Theorem 8, (3.18) holds for the first $n_B - (n_B \bmod k_{ub})$ data blocks of the movie. For practically relevant cases, n_B is much larger than k_{ub}. As a result, the last $(n_B \bmod k_{ub})$ data blocks hardly affect the value of t_{avg}. Furthermore, the right-hand side of (3.18) is not decreased by these data blocks. Hence, if k_{ub} is not a divisor of n_B, then (3.18) wil still approximately be true. As a result, whenever the transfer time that is reserved for a user in a data block is

smaller than the transfer time that on average is required for transferring a data block, where the average is taken over all data blocks, then no assignment exists such that (3.9) holds. Consequently, our approach is only applicable if $t_d \geq t_{\text{avg}}$.

Example 1 revisited. We showed that (3.9) holds for $k_{\text{ub}} = 6$. Consequently, by Theorem 8, $t_d \geq t_{\text{avg}}$ has to hold. We show that this is indeed the case. Because all zones are of equal size and because from the first six data blocks exactly one data block is stored in each zone, t_{avg} is given by the sum of the transfer times of these data blocks divided by six. Hence,

$$t_{\text{avg}} = \frac{\sum_{i=0}^{5} t_a(i)}{6} = \frac{22.27 + 20.55 + 19.09 + 17.81 + 16.70 + 15.72}{6} = 18.69. \tag{3.19}$$

As a result, $t_d = t_{\text{avg}}$. □

Because both t_d and t_{avg} depend on the block size and the maximum number of admitted users, the condition whether $t_d \geq t_{\text{avg}}$ can only be checked for given values of B and n. The following theorem states that this condition is equivalent to the condition that the block size must be based on a transfer rate that is at most the average transfer rate. For Theorem 1 this means that a data block has to be large enough to survive a sweep in which a data block is fetched for each user at an average transfer rate, i.e., $B \geq c_{\max}(n \cdot \frac{B}{r_{\text{avg}}} + s(n))$. Again, the block size is minimal when equality holds. With these conditions it can be determined for a given value of n for which block sizes our approach possibly gives a feasible assignment. Hence, we do not have to check for each combination of B and n whether $t_d \geq t_{\text{avg}}$ when you are interested in the buffer requirements for different values of B and n, but we only have to calculate the minimum block size for each value of n.

Theorem 9. *The condition $t_d \geq t_{\text{avg}}$ is equivalent to*

$$B \geq c_{\max}(n \cdot \frac{B}{r_{\text{avg}}} + s(n)) \tag{3.20}$$

in the case that the buffer requirements are given by Theorem 1 and to

$$B \geq c_{\max}(n \cdot \frac{B}{r_{\text{avg}}} + \frac{1}{2}s_3(2n)) \tag{3.21}$$

in the case that the buffer requirements are given by Theorem 3 and to

$$B \geq c_{\max}(n \cdot \frac{B}{r_{\text{avg}}} + s_2(n)) \tag{3.22}$$

in the case that the buffer requirements are given by Theorem 4.

Proof. We only prove the first equivalence. The other two can be proved similarly. As stated in Theorem 1, t_d satisfies $B = c_{\max}(n \cdot t_d + s(n))$, by definition. Consequently, Equation (3.20) is equivalent to $t_d \geq \frac{B}{r_{\text{avg}}}$. As a result, to prove the theorem is suffices to prove that $t_{\text{avg}} = \frac{B}{r_{\text{avg}}}$.

By definition, r_{avg} is the total amount of data that is stored on disk divided by the time required for transferring all these data. Hence,

$$r_{\text{avg}} = \frac{|F| \cdot B}{\sum_{i \in F} t(i)}.$$

Furthermore, $t_{\text{avg}} = \frac{\sum_{i \in F} t(i)}{|F|}$. Hence, $t_{\text{avg}} = \frac{B}{r_{\text{avg}}}$ can be written as

$$\frac{\sum_{i \in F} t(i)}{|F|} = \frac{B}{\frac{|F| \cdot B}{\sum_{i \in F} t(i)}},$$

which is clearly true. □

Chapter 4

Solution strategy

In the previous chapter we presented Algorithm 1 for minimizing the required buffer capacities given by Theorems 1 and 4 and for finding an assignment such that condition (2.10) holds in Theorem 3. For the algorithm, we still have to solve the problem of minimizing (3.17) for a given k_{ub}, $2 \leq k_{ub} \leq n_B$. We showed that this problem is NP-complete. Nevertheless, for the case that $k_{ub} = 2$, an optimal assignment can be defined. This assignment is given in Section 4.1. For the case that $k_{ub} > 2$, we give an heuristic in Section 4.2.

4.1 Window size of two

In the first iteration of Algorithm 1, (3.17) has to be minimized for the case that $k_{ub} = 2$. In this section we define the assignment a_2 that gives an optimal solution of the problem. As mentioned in the previous chapter, the problem of finding an assignment that satisfies (2.10) in Theorem 3 is solved by the first iteration of Algorithm 1. Consequently, this problem is optimally solved, i.e., there exists an assignment such that (2.10) holds if and only if a_2 satisfies (2.10).

Before we define a_2, we introduce some notation. We let Expression (3.17) for $k_{ub} = 2$ be denoted by $f_2(a)$ Hence,

$$f_2(a) = \max_{0 \leq i \leq n_B - 2} (t_a(i) + t_a(i+1)).$$

Furthermore, we define $w_a(i)$ as $t_a(i) + t_a(i+1)$, i.e., the sum of the transfer time of data block i and its successor.

We define a_2 as follows

$$a_2(i) = \begin{cases} \frac{i}{2} & \text{if } i \bmod 2 = 0 \\ n_B - 1 - \frac{i-1}{2} & \text{if } i \bmod 2 = 1. \end{cases} \tag{4.1}$$

Hence, for $i = 0, 1, 2, 3, \ldots$ the sequence values for a_2 are $0, n_B - 1, 1, n_B - 2, \ldots$, i.e., the data blocks are placed alternately on the innermost and outermost free positions of the disk. The sequence of a_2 is visualized in Figure 4.1. Before we prove that a_2 gives the optimal solution with regard to the function f_2, we prove a lemma. In terms of the sequence of a_2, the lemma states the following. When the first place of the sequence is denoted by place 0, then the value occurring at an odd place in the sequence of a_2 is followed by values that are all smaller than this value. Furthermore, the value at an even place in the sequence is followed by only larger values.

Lemma 2. *Let a_2 be the assignment given by (4.1). Then for each pair i, j, where $0 \leq i \leq j < n_B$, we have*

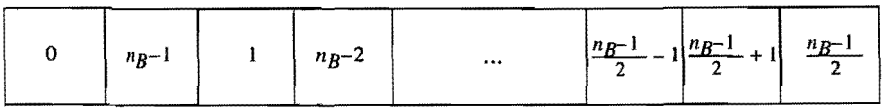

Figure 4.1. Assignment a_2 for the case that $n_B - 1$ is even

that

$$i \bmod 2 = 1 \Rightarrow a_2(j) \leq a_2(i) \tag{4.2}$$

and

$$i \bmod 2 = 0 \Rightarrow a_2(j) \geq a_2(i) \tag{4.3}$$

Proof. We first prove that (4.2) holds. Let i and j be two arbitrary blocks, such that $0 \leq i \leq j < n_B$ and $i \bmod 2 = 1$. We have to prove that $a_2(j) \leq a_2(i)$. From the definition of a_2, it follows that $a_2(j) \leq a_2(i)$ is equivalent to $n_B - 1 - \frac{j-1}{2} \leq n_B - 1 - \frac{i-1}{2}$ if $j \bmod 2 = 1$ and to $\frac{j}{2} \leq n_B - 1 - \frac{i-1}{2}$ otherwise. The first equation is implied by $i \leq j$. The second equation can be rewritten to $j + i \leq 2n_B - 1$. This is true because i and j are at most $n_B - 1$. This completes the proof of (4.2).

Next, we prove (4.3). Let i and j be two arbitrary blocks, such that $0 \leq i \leq j < n_B$ and $i \bmod 2 = 0$. We have to prove that $a_2(j) \geq a_2(i)$. By the definition of a_2, this is equivalent to $\frac{j}{2} \geq \frac{i}{2}$ if $j \bmod 2 = 0$ and to $n_B - 1 - \frac{j-1}{2} \geq \frac{i}{2}$, otherwise. The first equation follows directly from $i \leq j$. The second equation can be rewritten to $j + i \leq 2n_B - 1$. As above, this is true since $i, j \leq n_B - 1$. □

We can now prove the following theorem, that states the optimality of a_2.

Theorem 10. *Let a be an arbitrary assignment and a_2 the assignment given by (4.1). In that case, $f_2(a_2) \leq f_2(a)$.*

Proof. We will prove that, when an arbitrary assignment a, $a \neq a_2$, is given, there always exists an assignment a' satisfying

$$f_2(a') \leq f_2(a) \wedge \min\{i \in L | a(i) \neq a_2(i)\} < \min\{i \in L | a'(i) \neq a_2(i)\}. \tag{4.4}$$

This means, that we will prove the existence of an assignment a' with at most the same value for f_2 as a and with the property that the number of the first block where a and a_2 differ, is smaller than the number of the first block where a' and a_2 differ. It is easy to see that this proves the theorem.

Let a be an arbitrary assignment, different from a_2. We will prove that an assignment a' that satisfies (4.4) exists by defining it. Define m as the first block, where a and a_2 differ. Formally, $m = \min\{i \in L | a'(i) \neq a_2(i)\}$. Moreover, let p be the block that has been assigned to place $a_2(m)$ by assignment a. We now distinguish the following two cases.

1. $m \bmod 2 = 0$. Define a' as

$$a'(i) = \begin{cases} a(i) & \text{if } 0 \leq i < m \\ a(p+i-m) & \text{if } m \leq i < m + n_B - p \\ a(n_B + m - i - 1) & \text{if } m + n_B - p \leq i < n_B. \end{cases} \tag{4.5}$$

2. $m \bmod 2 = 1$. Define a' as

$$a'(i) = \begin{cases} a(i) & \text{if } 0 \leq i < m \\ a(p+i-m) & \text{if } m \leq i < m + n_B - p \\ a(i - n_B + p) & \text{if } m + n_B - p \leq i < n_B. \end{cases} \tag{4.6}$$

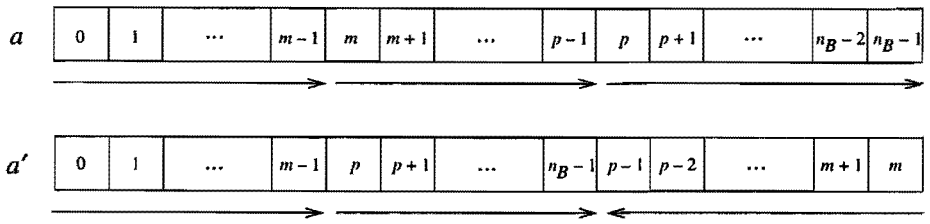

Figure 4.2. Definition (4.5)

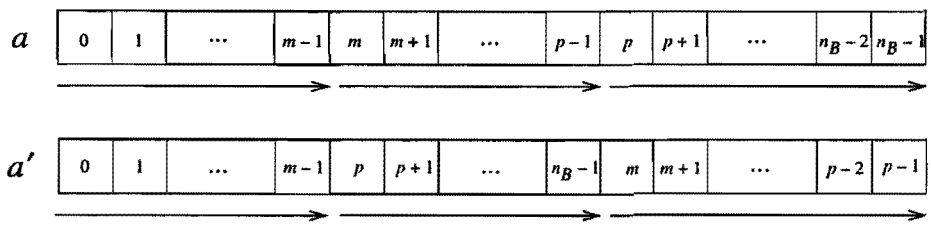

Figure 4.3. Definition (4.6)

These two definitions are depicted in Figures 4.2 and 4.3, respectively. We first show, by contradiction, that $p > m$. So, assume that $p \leq m$. From the definitions of m and p it follows that p can not be equal to m. What remains is the case $p < m$. Because, by definition, $a(i) = a_2(i)$ for all i from 0 through $m-1$, $a(p) = a_2(p)$ holds. Furthermore, we have, by definition, $a(p) = a_2(m)$. Hence, $a_2(p) = a_2(m)$ and $p < m$. But this is in contradiction with the bijectivity of a_2. Hence, $p > m$.

Consider the case that a' is defined according to (4.5). We will prove that (4.4) holds. In terms of its sequence, the definition comes down to appending the last $n_B - p$ elements to the first $m-1$ elements and appending the elements m through $p-1$ in reverse order to the end of the sequence. We first prove that $f_2(a')$ is at most $f_2(a)$. By definition, $f_2(a') = \max_{i \in L \setminus \{n_B-1\}} w_{a'}(i)$ and $f_2(a) = \max_{i \in L \setminus \{n_B-1\}} w_a(i)$. It can be verified that the only terms that occur in $f_2(a')$ and not in $f_2(a)$ are $w_{a'}(m-1)$ and $w_{a'}(m+n_B-p-1)$. Expressed in a, these terms are $t_a(m-1)+t_a(p)$ and $t_a(n_B-1)+t_a(p-1)$, respectively. To prove $f_2(a') \leq f_2(a)$, it suffices to prove that these two terms are at most $f_2(a)$.

Consider the term $t_a(m-1)+t_a(p)$. Assume that $t_a(m-1) \leq t_a(p-1)$. This implies that $t_a(m-1)+t_a(p) \leq w_a(p)$. From the definition of f_2, it now follows that $t_a(m-1)+t_a(p) \leq f_2(a)$. Hence, it suffices to prove that $t_a(m-1) \leq t_a(p-1)$. We do this by contradiction. So, assume that $t_a(p-1) < t_a(m-1)$. Since t is descending, $a(p-1) > a(m-1)$. Because a and a_2 have the same range and because $a(m-1) = a_2(m-1)$, as follows from the definition of m, we can rewrite this equation to $a_2(j) > a_2(m-1)$, where $a_2(j) = a_2(p-1)$. From (4.2) and from $(m-1) \bmod 2 = 1$, which is true because of the assumption $(m \bmod 2 = 0)$, it follows that j has to be strictly smaller than $m-1$. But because of the definition of m, this implies that $a(j) = a_2(j)$. Hence, $a(j) = a(p-1)$, where $j < m-1$ and $p > m$. This is in contradiction with the bijectivity of a.

Next, we prove that $t_a(n_B-1)$ is at most $t_a(p)$. Similar as above, this suffices to prove that $t_a(n_B-1)+t_a(p-1) \leq f_2(a)$. Again, the proof is by contradiction. So, suppose that $t_a(p) < t_a(n_B-1)$. Since t is descending, $a(n_B-1) < a(p)$. Because a and a_2 have the same range and because $a(p) = a_2(m)$, we can rewrite this equation to $a_2(j) < a_2(m)$, where $a_2(j) = a(n_B-1)$. By assumption, we have that $(m \bmod 2 = 0)$. This and (4.3) yield that j has to be strictly smaller than m. Consequently, $a(j) = a_2(j)$. We now have $a(j) = a(n_B-1)$, where $j < m$ and $m \leq n_B - 1$. This is in contradiction with the bijectivity of a. This completes the proof of $f_2(a') \leq f_2(a)$.

From the definitions of m and p it follows that $(\min_{i \in L} a(i) \neq a_2(i)) < (\min_{i \in L} a'(i) \neq a_2(i))$. Consequently, a' satisfies (4.4), in the case that it is defined according to (4.5).

Consider the case that a' is defined according to (4.6). This definition comes down to appending the last

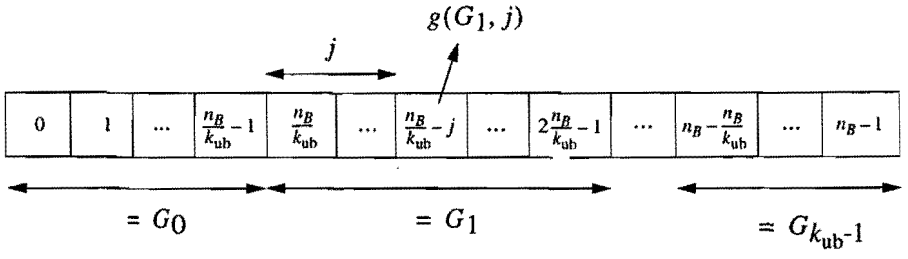

Figure 4.4. Definitions of G_i and g for the case that $k_{ub}|n_B$.

$n_B - p$ elements to the first m elements and appending the elements m through $p-1$ to the end of the sequence. Again, we start with the proof of $f_2(a') \leq f_2(a)$. The only terms of $f_2(a')$ that are absent in $f_2(a)$ are $w_{a'}(m-1)$ and $w_{a'}(m+n_B-p-1)$. These terms equal $t_a(m-1)+t_a(p)$ and $t_a(n_B-1)+t_a(m)$, respectively. We prove that these terms are at most $f_2(a)$.

Consider $t_a(m-1)+t_a(p)$. We prove that this term is at most $f_2(a)$, by proving that it is at most $w_a(m-1)$. Hence, we have to show that $t_a(p) \leq t_a(m)$. Suppose that $t_a(p) > t_a(m)$. We prove that this leads to a contradiction. Since t is descending $a(p) < a(m)$. This can be rewritten to $a_2(m) < a_2(j)$, where $a_2(j) = a(m)$, because of the definition of p and because a and a_2 have the same range. By assumption, we have that $(m \bmod 2 = 1)$. From (4.2) can now be concluded that $j < m$. Consequently, $a(j) = a_2(j)$. We now have that $a(j) = a(m)$ and $j < m$. This is in contradiction with the bijectivity of a.

We now prove that $t_a(n_B-1)+t_a(m)$ is at most $w_a(m-1)$, as well. This means, that we have to prove that $t_a(n_B-1) \leq t_a(m-1)$. We do this by showing that the assumption that $t_a(n_B-1) > t_a(m-1)$ leads to a contradiction. The equation $t_a(n_B-1) > t_a(m-1)$ implies $a(n_B-1) < a(m-1)$. When we write this in terms of a_2 we get that $a_2(j) < a_2(m-1)$, where $a_2(j) = a(n_B-1)$. From the assumption that $(m \bmod 2 = 1)$ and from (4.3) it follows that $j < m-1$. Hence, $a(j) = a_2(j)$. Because, by definition, $a_2(j) = a(n_B-1)$, we have $a(j) = a(n_B-1)$. The equations $j < m-1$ and $m \leq n_B - 1$ and the bijectivity of a now yield the contradiction.

Again, it is easy to see that $(\min_{i \in L} a(i) \neq a_2(i)) < (\min_{i \in L} a'(i) \neq a_2(i))$. Hence, a' also satisfies (4.4), in the case that is defined according to (4.6). □

4.2 Heuristic for windowsize at least three

In this section, we present a heuristic for minimizing (3.17) for the case that $k_{ub} \geq 3$. By Theorem 8, $k_{ub} \cdot t_{avg}$ is a lowerbound for the value of (3.17). Hence, it is our objective to get as close to $k_{ub} \cdot t_{avg}$ as possible.

We first give a global idea of the heuristic. The set of positions, i.e., F, is divided into k_{ub} more or less equal-sized sets, such that the positions of each set are contiguous. For the case that k_{ub} is a divisor of n_B these sets, denoted by G_0 through $G_{k_{ub}-1}$, are depicted in Figure 4.4. Because the positions of each set are contiguous, the average transfer time of an arbitrary subsequence of length k_{ub} will be close to t_{avg} if assignment a is defined such that each subsequence of length k_{ub} contains exactly one value from each set. Therefore, the positions from the same set are placed k_{ub} places from each other when defining the sequence of a.

As discussed in Section 3.2, we minimize (3.17) in order to find an assignment for which (3.9) holds, because then, by Theorem 6, the buffer size is determined by the maximum cumulative difference between transfer time and dimension time of any subsequence of length at most k_{ub}. Consequently, in addition to minimizing (3.17), we aim at an assignment in which this cumulative difference is minimized. Therefore, we define an ordering of the sets G_i, $0 \leq i < k_{ub}$, that determines in which order the positions from each set occur in the sequence of a and correspondingly in each subsequence of length at most k_{ub}. This ordering has to be such that sets that contain positions with a relatively low transfer rate are alternated with sets that

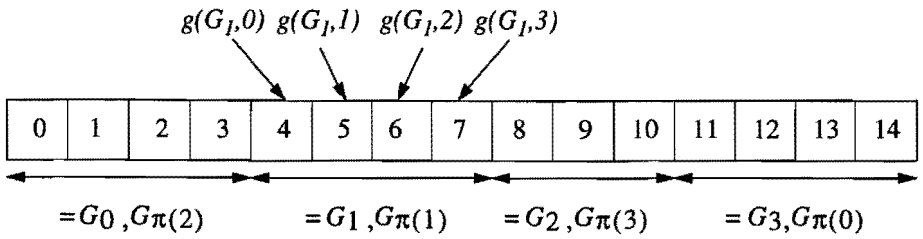

Figure 4.5. Definitions of G_i and g in Example 4

contain relatively high transfer times.

We now formalize the heuristic. We start with the introduction of some definitions. As mentioned, we divide the set of positions of the hard disk into k_{ub} disjoint sets, denoted by G_0 through $G_{k_{\text{ub}}-1}$. Moreover, we let π be a permutation of $\{0,1,\ldots,k_{\text{ub}}-1\}$ indicating the ordering of the sets. If k_{ub} is a divisor of n_B, then we let the set G_0 contain the inner $\frac{n_B}{k_{\text{ub}}}$ positions of the hard disk, G_1 the next $\frac{n_B}{k_{\text{ub}}}$ positions, and so on (see Figure 4.4). We define the sets similarly in the case that k_{ub} is not a divisor of n_B, except that we let the first ($n_B \bmod k_{\text{ub}}$) sets, i.e., the sets $G_{\pi(0)}$ through $G_{\pi(n_B \bmod k_{\text{ub}}-1)}$, contain one element more than the others. We define the set W as the indices of these sets. Hence, $W = \{i|(\exists_{x<n_B \bmod k_{\text{ub}}}\pi(x) = i)\}$. Note that if k_{ub} is a divisor of n_B, then $W = \emptyset$, which corresponds to its meaning.

To give a formal definition of the sets G_0 through $G_{k_{\text{ub}}-1}$, we investigate which position is the first position that an arbitrary set G_i contains. In the case that k_{ub} is a divisor of n_B, the first position that is in G_i is $i \cdot \frac{n_B}{k_{\text{ub}}}$, because each set has exactly $\frac{n_B}{k_{\text{ub}}}$ elements. If, on the other hand, k_{ub} is not a divisor of n_B, then a set G_j has $\lfloor\frac{n_B}{k_{\text{ub}}}\rfloor + 1$ elements if $j \in W$ and $\lfloor\frac{n_B}{k_{\text{ub}}}\rfloor$ otherwise. Consequently, the first position that is an element of G_i is given by $i \cdot \lfloor\frac{n_B}{k_{\text{ub}}}\rfloor + \Delta(i)$, where $\Delta(i)$ is the number of sets with a smaller index than i and that are an element of W. Hence, $\Delta(i) = |\{j \in W | j < i\}|$. If k_{ub} is a divisor of n_B, then the first position that is an element of G_i can also be written as $i \cdot \lfloor\frac{n_B}{k_{\text{ub}}}\rfloor + \Delta(i)$, because $\Delta(i) = 0$ for all i, $0 \leq i < k_{\text{ub}}$. Because the cardinality of G_i is in both cases given by $\lfloor\frac{n_B}{k_{\text{ub}}}\rfloor + \chi_{(W)}(i)$, this yields that G_i is defined by

$$\{i \cdot \lfloor\frac{n_B}{k_{\text{ub}}}\rfloor + \Delta(i) + j \mid 0 \leq j < \lfloor\frac{n_B}{k_{\text{ub}}}\rfloor + \chi_{(W)}(i)\}.$$

We define the function g, such that $g(G_i, j)$ gives the position for which it holds that the set G_i has exactly j positions with a smaller number. Because t is a descending function, this means that $g(G_i, j)$ gives the position from G_i for which it holds that there are at most j positions in G_i with a higher transfer time. Formally, $g(G_i, j) = i \cdot \lfloor\frac{n_B}{k}\rfloor + \Delta(i) + j$, where $0 \leq i < k_{\text{ub}}$ and $0 \leq j < |G_i|$ (see Figure 4.4). From the definitions of G and g, it follows that $g(G_i, j)$ is increasing in i and for fixed i also in j. Since t is descending, this is implies that $t(g(G_i, j))$ is descending in i and for fixed i in j, i.e.,

$$i_0 < i_1 \Rightarrow t(g(G_{i_0}, j_0)) \geq t(g(G_{i_1}, j_1)), \text{ for all } j_0, j_1 \tag{4.7}$$

and

$$i_0 = i_1 \wedge j_0 < j_1 \Rightarrow t(g(G_{i_0}, j_0)) \geq t(g(G_{i_1}, j_1)). \tag{4.8}$$

Example 4. Let the disk consist of 15 positions and let k_{ub} be 4. Hence, the positions are divided into four sets, G_0, G_1, G_2 and G_3. We define π as $(3,1,0,2)$. This means that the ordering of the sets is G_3, G_1, G_0, G_2. Consequently, $W = \{i|\exists_{x<3}\pi(x) = i\} = \{0,1,3\}$. As a result, the sets G_0, G_1 and G_3 contain four contiguous elements and G_2 only three. Figure 4.5 gives the definition of the sets and depicts the function g for the set G_1. □

We now discuss the construction of assignment a. For the time being we let π define an arbitrary ordering because π does not affect the discussion about the performance of the heuristic in relation to minimizing (3.17). At the end of the section, we briefly discuss the choice of π. As discussed, we define assignment

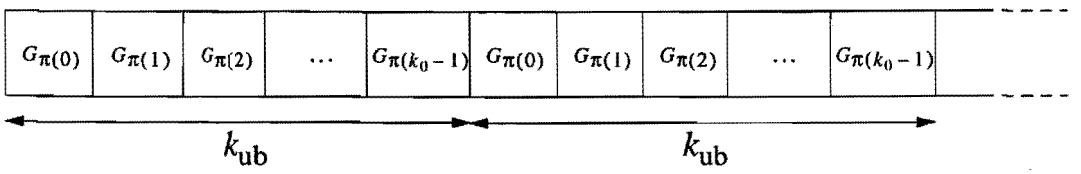

Figure 4.6. Outline of assignment a

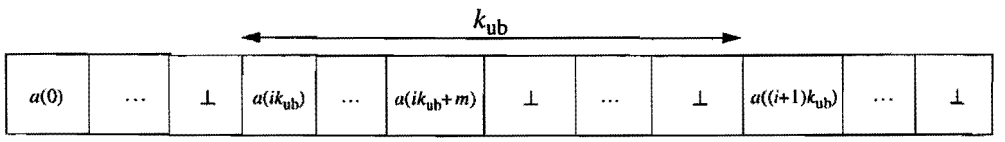

Figure 4.7. Assignment a after $m+1$ iteration steps, where $\perp$ stands for an undefined value

a, such that each subsequence of length k_{ub} contains exactly one value from each set G_i, $0 \leq i < k_{ub}$. To achieve this, the definition of the assignment is as depicted in Figure 4.6. Hence, data blocks whose numbers differ a multiple of k_{ub} are assigned to positions from the same set. Formally, a is defined, such that data block j is assigned to a position from set $\pi(j \bmod k_{ub})$. Because a has to be a bijection, no position may occur twice in the sequence of a. It follows from the definition of G_i, $0 \leq i < k_{ub}$, that each set contains enough positions to make this possible.

For each data block i, we are still free to choose to which position from $G_{\pi(i \bmod k_{ub})}$ it is to be assigned. We discuss this choice in more detail. We denote the set of data blocks that have to be assigned to the positions from $G_{\pi(m)}$ by V_m. By definition, two data blocks are assigned to positions from the same set if and only if their numbers differ a multiple of k_{ub}. Hence, V_m is given by

$$V_m = \{i \cdot k_{ub} + m | 0 \leq i < \lfloor \frac{n_B}{k_{ub}} \rfloor + \chi_{(W)}(m)\}.$$

We define assignment a in a stepwise way. Initially, no positions are assigned to the data blocks, i.e., all places in the sequence of a are undefined. In the first step, denoted by iteration 0, the positions from the set $G_{\pi(0)}$ are assigned to the data blocks from V_0. Next, the positions from $G_{\pi(1)}$ are assigned to the data blocks from V_1, and so on until the positions from $G_{\pi(k_{ub}-1)}$ are assigned to the data blocks from $V_{k_{ub}-1}$. Consequently, after iteration m each subsequence of a of length k_{ub} contains exactly one value from each of the sets $G_{\pi(0)}$ through $G_{\pi(m)}$ (see Figure 4.7). We aim at an assignment for which the maximum sum of the transfer times of k_{ub} successive data blocks is minimal. Therefore, we aim in iteration m at assigning the positions from $G_{\pi(m)}$ to the data blocks from V_m in such a way that, as far as the sequence is defined, the maximum sum of the transfer times over all subsequences of length k_{ub} is minimal. By definition, each subsequence of length k_{ub} contains exactly one data block from V_i, $0 \leq i < k_{ub}$. Consequently, our aim comes down to minimizing $\max_{i \in V_m} h^m(i)$, where $h^q(p)$ is the function that, as far as the subsequences are defined at the end of iteration q, gives the maximum sum of the transfer times over all subsequences of length k_{ub} that contain data block p. If $q = -1$, then, by definition, this means that the initial situation is considered, i.e., the moment that a is undefined over its entire domain. Consequently, $h^{-1}(i) = 0$ for all $i \in L$. Because the subsequences that contain data block p are the subsequences that have a data block between $p - k_{ub} + 1$ and p as their first data block, this means that

$$h^q(p) = \max_{\substack{0 \leq i \leq n_B - k_{ub} \\ p - k_{ub} < i \leq p}} \sum_{\substack{j=0 \\ a(i+j) \text{ is defined} \\ \text{after iteration } q}}^{k_{ub}-1} t_a(i+j).$$

In the next lemma, we prove that during iteration m, $0 \leq m < k_{ub}$, i.e., the step in the construction of a in which the positions from $G_{\pi(m)}$ are assigned to the data blocks from V_m, the value of $h^m(i)$, $i \in V_m$, is exactly

Algorithm 2 Placing algorithm.

Define π and G_0 through $G_{k_{ub}-1}$.
forall $0 \leq m < k_{ub}$
  $\omega_m :=$ sequence containing the values from V_m in order of descending h^{m-1}-value;
  forall $0 \leq i < |G_{\pi(m)}|$
    $a(\omega_m(i)) := g(G_{\pi(m)}, |G_{\pi(m)}| - 1 - i)$;

$t_a(i)$ larger than $h^{m-1}(i)$. In other words, the maximum sum of the transfer times over all subsequences that contain data bock i is raised by exactly the transfer time required for transferring data block i from position $a(i)$.

Lemma 3. *Let* $0 \leq m < k_{ub}$. *Then*

$$(\forall_{i \in V_m} h^m(i) = h^{m-1}(i) + t_a(i)).$$

Proof. Let m be arbitrary. By definition, the value of $h^{m-1}(i)$, $i \in V_m$, is the maximum sum of transfer times over all subsequences of a that contain data block i, as far as the subsequences are defined at the end of iteration $m-1$. Because $a(i)$ is undefined at the start of iteration m and is defined during this step, the sum of the transfer times of each subsequence that contains data block i is raised by at least $t_a(i)$ during iteration m. Hence, $h^m(i) \geq h^{m-1}(i) + t_a(i)$. Because a is constructed, such that each subsequence of length k_{ub} contains exactly one element from V_m, i is the only data block to which a position is assigned in iteration m in those subsequences. Hence, $h^m(i) = h^{m-1}(i) + t_a(i)$. □

Consider iteration m. Let ω_m be the sequence containing the numbers from V_m in order of descending value of h^{m-1}. By Lemma 3, minimizing $\max_{i \in V_m} h^m(i)$ comes down to assigning the positions from $G_{\pi(m)}$ with low transfer times to data blocks for which the function value of h^{m-1} is high and vice versa. Furthermore, it follows from (4.8) that

$$t(g(G_{\pi(m)}, |G_{\pi(m)}| - 1)) \leq t(g(G_{\pi(m)}, |G_{\pi(m)}| - 2)) \leq \cdots \leq t(g(G_{\pi(m)}, 0)).$$

Hence, $\max_{i \in V_m} h^m(i)$ is minimized in iteration m, if

$$a(\omega_m(0)) = g(G_{\pi(m)}, |G_{\pi(m)}| - 1), a(\omega_m(1)) = g(G_{\pi(m)}, |G_{\pi(m)}| - 2), \ldots.$$

Hence, if $a(\omega_m(i)) = g(G_{\pi(m)}, |G_{\pi(m)}| - 1 - i)$. Algorithm 2 depicts the discussed algorithm.

Example 4 revisited. Let the transfer time of a position be one plus its number. Hence, $t(i) = i + 1$, $0 \leq i < 15$. In the first iteration of Algorithm 2, the positions of $G_{\pi(0)}$ $(= G_3)$ are placed in the sequence of a. Because all places in the sequence of a are initially undefined, $h^{-1}(i) = 0$ for all $i \in V_0$, where, by definition, $V_0 = \{0, 4, 8, 11\}$. Hence, the sequence ω_0 may be arbitrary. When we take the sequence 11 8 4 0 as ω_0, we get the sequence given in Figure 4.8 after assigning the positions from $G_{\pi(0)}$ to the data blocks from V_0. Note that Figure 4.8 depicts t_a.

In the second iteration the positions from G_1 are assigned to the data blocks from $V_1 = \{1, 5, 9, 12\}$. Hence, we have to determine $h^0(1)$, $h^0(5)$, $h^0(9)$ and $h^0(12)$. The subsequence of length four, containing data block 1 with the highest sum of the transfer times, is the subsequence starting at data block 1 and ending at data block 4 and has as value 13. Consequently, $h^0(1) = 13$. Similarly, it can be verified that $h^0(5) = 14$, $h^0(9) = 15$ and $h^0(12) = 15$. As a result, ω_1 is either 12 9 5 1 or 9 12 5 1. Figure 4.8 depicts the result of assinging the values from G_1 to the data blocks for the case that ω_1 is defined by the first sequence. The result of the last two iterations are also depicted in Figure 4.8. From the figure it can be read that the value of Expression (3.17) for the obtained sequence is 33. As mentioned, Theorem 8 is true if $k_{ub} | n_B$ and is approximately true, otherwise. Because $4 \not| 15$, this yields that it will not be possible to define an assignment such that (3.17) is much smaller than $k_{ub} \cdot t_{avg}$. It is easily verified that $t_{avg} = 8$. Hence, $k_{ub} \cdot t_{avg} = 32$. Consequently, the value of (3.17) for the obtained assignment is close to the optimal value

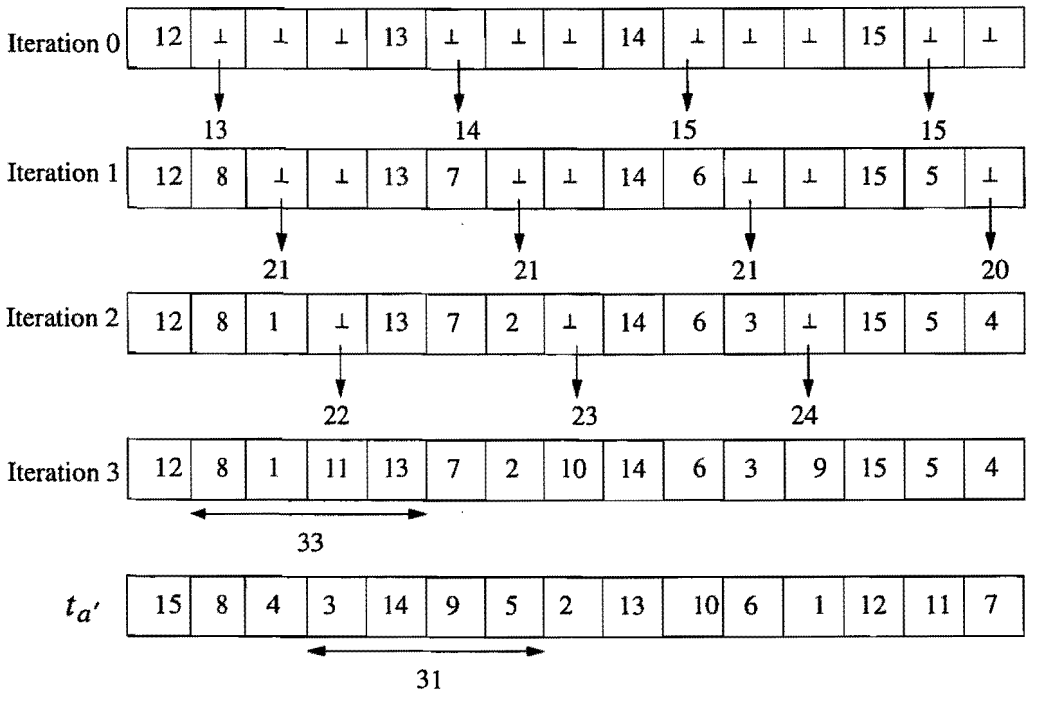

Figure 4.8. Example of the execution of Algorithm 2

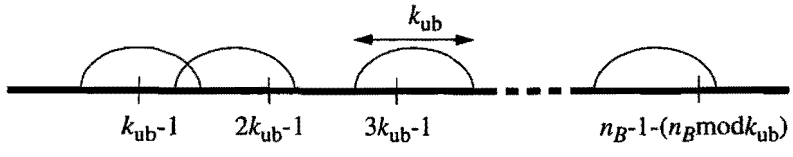

Figure 4.9. Interpretation of Lemma 4.

of (3.17).

In Figure 4.8, we also give the optimal assignment a'. For a' Expression (3.17) equals 31. □

In the last iteration step positions are assigned to the data blocks from the set $V_{k_{ub}-1}$. In terms of the sequence of a this comes down to assigning positions to the places $k_{ub}-1, 2k_{ub}-1, \ldots, n_B-1-(n_B \bmod k_{ub})$. In the next lemma we prove that each of these places is in a subsequence of length k_{ub} for which it holds that the sum of the transfer times differs at most α from the maximum sum of the transfer times over all subsequences of length k_{ub}, i.e., from the value of (3.17), where α is the maximum difference in transfer time between the positions of the same set G_m, $0 \leq m < k_{ub}$. Formally, for all $p \in V_{k_{ub}}$

$$\max_{0\leq i\leq n_B-k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j) - \alpha \leq h^{k_{ub}}(p) \leq \max_{0\leq i\leq n_B-k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j). \tag{4.9}$$

By definition, $\max_{i\in V_{k_{ub}-1}} h^{k_{ub}-1}(i) = \max_{0\leq i\leq n_B-k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j)$. Consequently, (4.9) holds for all $p \in V_{k_{ub}-1}$ if and only if $|h^{k_{ub}-1}(p_1) - h^{k_{ub}-1}(p_2)| \leq \alpha$ for all $p_1, p_2 \in V_{k_{ub}-1}$. Figure 4.9 visualizes the property and is to be interpreted as follows. An arc is drawn over each place corresponding to a data block $i \in V_{k_{ub}-1}$ and its position and indicates the subsequence of length k_{ub} that contains data block i and for which the sum of the transfer times is maximal. The property implies that the sum of the transfer times of two arcs differ at most α.

Because of (4.8) the definition of α is given by

$$\alpha = \max_{0 \leq m < k_{ub}} (t(g(G_m, 0)) - t(g(G_m, |G_m| - 1))). \tag{4.10}$$

Because the number of zones on a disk are relatively small (the Seagate Barracuda 9 drive consists of 7 zones as follows from Table 1.1), the value of α is for already for small values of k_{ub} equal to the maximum difference in transfer time between two successive zones.

Lemma 4. *Let $p_1, p_2 \in V_{k_{ub}-1}$ and a an assignment that is constructed by Algorithm 2. Then*

$$|h^{k_{ub}-1}(p_1) - h^{k_{ub}-1}(p_2)| \leq \alpha \tag{4.11}$$

where α is defined by (4.10).

Proof. We prove (4.11) by proving by induction to m that for all $p_1, p_2 \in V_m$,

$$|h^m(p_1) - h^m(p_2)| \leq \alpha. \tag{4.12}$$

As basis case we take $m = 0$. By definition, $h^{-1} = 0$ for all $i \in V_0$. Hence, by Lemma 3, $h^1(i) = t_a(i)$. During the iteration, only places from $G_{\pi(0)}$ are assigned to the sequence. Hence, by the definition of α it holds that for all $i, j \in V_0$ $|h^0(i) - h^0(j)| \leq \alpha$. Consequently, (4.12) holds in the case that $m = 0$.

Next, we prove (4.12) for $m+1$, $m \geq 0$. Let $p_1, p_2 \in V_{m+1}$. Assume that $|h^m(p_1) - h^m(p_2)| \leq \alpha$. Without loss of generality, we may assume that $h^m(p_1) \geq h^m(p_2)$. By construction, the position that is assigned to data block p_1 during iteration $m+1$ has at most the same transfer time than the position that is assigned to data block p_2. Furthermore, by the definition of α, we have that these transfer times differ at most α. By Lemma 3, $h^{m+1}(i_1)$ and $h^{m+1}(i_2)$ are raised by $t_a(p_1)$ and $t_a(p_2)$ in relation to $h^m(p_1)$ and $h^m(p_2)$, respectively. As a result, $|h^{m+1}(p_1) - h^{m+1}(p_2)| \leq \alpha$.

Hence, if $|h^m(p_1) - h^m(p_2)| \leq \alpha$, then $|h^{m+1}(p_1) - h^{m+1}(p_2)| \leq \alpha$, which has to be proved. By assumption, $h^m(p_1) \geq h^m(p_2)$ and $p_1 \in V_{m+1}$. Consequently, $|h^m(p_1) - h^m(p_2)| \leq \alpha$ is implied by

$$\max_{j \in V_{m+1}} h^m(j) - h^m(p_2) \leq \alpha. \tag{4.13}$$

By definition, $\max_{i \in V_{m+1}} h^m(i) = \max_{i \in V_m} h^m(i)$. Hence, (4.13) is equivalent to

$$\max_{j \in V_m} h^m(j) - h^m(p_2) \leq \alpha. \tag{4.14}$$

By the induction hypothesis, $|h^m(i_1) - h^m(i_2)| \leq \alpha$ for all $i_1, i_2 \in V_m$. By definition, $p_2 \in V_{m+1}$ implies $p_2 - 1 \in V_m$. Hence, $\max_{j \in V_m} h^m(j) - h^m(p_2 - 1) \leq \alpha$. Consequently, (4.14) is implied by $h^m(p_2 - 1) \leq h^m(p_2)$.

It can be read from the definition of h that the only differences between $h^m(p_2 - 1)$ and $h^m(p_2)$ are that if $p_2 \geq k_{ub}$, then $p_2 - k_{ub}$ is an element of the range of the maximum quantification of $h^m(p_2 - 1)$ and not of $h(p_2)$ and that if $p_2 \leq n_B - k_{ub}$, then p_2 is an element of the maximum quantification of $h^m(p_2)$ and not of $h^m(p_2 - 1)$.

The above-mentioned has the following interpretation. If $k_{ub} \leq p_2$, then there is exactly one subsequence of a that contains the $(p_2 - 1)$th place of the sequence and not the p_2th, namely the subsequence that starts with data block $p' - k_{ub}$, and there is no such sequence otherwise. Similarly, we have that if $p_2 \leq n_B - k_{ub}$, then the subsequence that starts with data block p_2 is the only subsequence that contains the data block p_2, but not $p_2 - 1$ and else such a sequence does not exist.

This implies that for proving that $h^m(p_2 - 1) \leq h^m(p_2)$, we only have to show that if $p_2 \geq k_{ub}$, then the sum of the transfer times of the subsequence of a that starts at data block $p_2 - k_{ub}$ is at most $h^m(p_2)$, where the subsequence is considered at the end of iteration m. Formally, this means that we have to prove that

$$\sum_{\substack{j=0 \\ a(p_2 - k_{ub} + j) \text{ is defined}}}^{k_{ub}-1} t_a(p_2 - k_{ub} + j) \leq h^m(p_2). \tag{4.15}$$

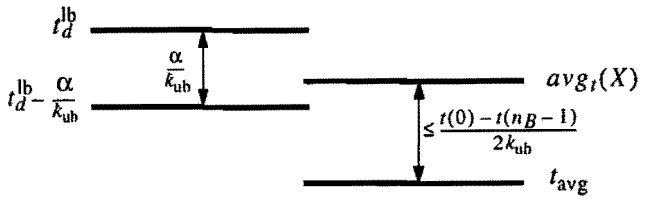

Figure 4.10. Outline of the proof of theorem 11

Consider the left-hand side of the equation. By definition we have that $p_2 \in V_{m+1}$. Hence, $a(p_2 - k_{ub})$ and $a(p_2)$ are both defined during the iteration $m+1$. Because the subsequence is considered the end of iteration m, this means that $a(p_2 - k_{ub})$ and $a(p_2)$ are both undefined. As a result, the sum of the transfer times of the subsequence that starts with data block $p_2 - k_{ub}$ equals the sum of the transfer times of the subsequence that starts with data block $p_2 - k_{ub} + 1$. Formally, this means that the left-hand side of (4.15) equals

$$\sum_{\substack{j=0 \\ a(p_2-k_{ub}+1+j) \text{ is defined}}}^{k_{ub}-1} t_a(p_2 - k_{ub} + 1 + j).$$

Because $p_2 - k_{ub} + 1$ is an element of the range of the maximum quantification in $h^m(p_2)$ we have that this expression is at most $h^m(p_2)$, which means that (4.15) holds. Hence, $h^m(p_2 - 1)$ is a lowerbound for $h^m(p_2)$. □

Using Lemma 4, we can prove the following theorem, which gives an upperbound on the value of (3.17) for an assignment constructed by Algorithm 2, in the case that k_{ub} is a divisor of n_B.

Theorem 11. *Let a be an assignment constructed by Algorithm 2 for a windowsize k_{ub} that is a divisor of n_B. Then*

$$\max_{0 \leq i \leq n_B - k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j) \leq k_{ub} \cdot t_{avg} + \alpha + \frac{t(0) - t(n_B - 1)}{2}, \tag{4.16}$$

where α is defined according to (4.10).

Proof. Dividing both sides in (4.16) by k_{ub} yields that proving (4.16) is equivalent to proving that

$$\frac{\max_{0 \leq i \leq n_B - k_{ub}} \sum_{j=0}^{k_{ub}-1} t_a(i+j)}{k_{ub}} \leq t_{avg} + \frac{\alpha}{k_{ub}} + \frac{t(0) - t(n_B - 1)}{2k_{ub}}. \tag{4.17}$$

As mentioned in Section 3.2.2, the left-hand side gives the lowerbound on the value of t_d for which (3.9) holds. Therefore, we denote the expression in the left-hand side by t_d^{lb}.

We define X as the set containing the $\lceil \frac{|G_i|}{2} \rceil$ $(= \lceil \frac{n_B}{2k_{ub}} \rceil)$ inner places of each set G_i, $0 \leq i < k_{ub}$. Hence,

$$X = \{g(G_i, j) | 0 \leq i < k_{ub} \wedge 0 \leq j < \lceil \frac{n_B}{2k_{ub}} \rceil\}$$

The proof strategy we will use to prove the theorem is outlined in Figure 4.10. We will first show that $avg_t(X) \geq t_d^{lb} - \frac{\alpha}{k_{ub}}$ and next that the difference between $avg_t(X)$ and t_{avg} is at most $\frac{t(0)-t(n_B-1)}{2k_{ub}}$. Because $\alpha \geq 0$, this proves (4.17).

Let $m \in V_{k_{ub}-1}$. By Lemma 4 or, equivalently, (4.9), there exists a subsequence of length k_{ub} that contains data block m, such that

$$t_d^{lb} - \frac{\alpha}{k_{ub}} \leq \frac{\sum_{i \in S_m} t(i)}{k_{ub}} \leq t_d^{lb},$$

where S_m is the set containing the positions in the subsequence. Because a subsequence is related to k_{ub} successive data blocks and because a is a bijection, S_m and $S_{m+2k_{\text{ub}}}$ are disjoint sets. Hence, there exist $\lceil |V_{k_{\text{ub}}-1}|/2 \rceil$ $(= \lceil n_B/2k_{\text{ub}} \rceil)$ sets, $S_{k_{\text{ub}}-1}, S_{3k_{\text{ub}}-1}, S_{5k_{\text{ub}}-1} \ldots$, that are all mutually disjoint and have a average transfer time between $t_d^{\text{lb}} - \frac{\alpha}{k_{\text{ub}}}$ and t_d^{lb}. Clearly, the average transfer time of the union of these sets is between these values, as well. We define the set Y as this union. Hence,

$$Y = \bigcup_{0 \le i < \lceil \frac{n_B}{2k_{\text{ub}}} \rceil} S_{k_{\text{ub}}-1+2ik_{\text{ub}}}$$

and $t_d^{\text{lb}} - \frac{\alpha}{k_{\text{ub}}} \le avg_t(Y) \le t_d^{\text{lb}}$. By construction, each subsequence of a of length k_{ub} contains one position from each set G_i, $0 \le i < k_{\text{ub}}$. Consequently, Y contains $\lceil \frac{n_B}{2k_{\text{ub}}} \rceil$ positions from each set G_i, $0 \le i < k_{\text{ub}}$. Because the set X contains the $\lceil \frac{n_B}{2k_{\text{ub}}} \rceil$ positions with the highest transfer rate, we obtain that $avg_t(Y) \le avg_t(X)$. This and $t_d^{\text{lb}} - \frac{\alpha}{k_{\text{ub}}} \le avg_t(Y)$ yields that $t_d^{\text{lb}} - \frac{\alpha}{k_{\text{ub}}} \le avg_t(X)$. This completes the first part of the proof.

The other part is

$$avg_t(X) \le t_{\text{avg}} + \frac{t(0) - t(n_B - 1)}{2k_{\text{ub}}}. \tag{4.18}$$

We first introduce a definition. We define G_i^{in} as the $\lceil \frac{n_B}{2k_{\text{ub}}} \rceil$ inner places of G_i, $0 \le i < k_{\text{ub}}$, and G_i^{out} as the $\lfloor \frac{n_B}{2k_{\text{ub}}} \rfloor$ outer places. Because $G_i^{\text{in}} \cup G_i^{\text{out}} = G_i$ and $G_i^{\text{in}} \cap G_i^{\text{out}} = \emptyset$, the value of t_{avg} is expressed by

$$\frac{\left(\sum_{i=0}^{k_{\text{ub}}-1} avg_t(G_i^{\text{in}})\right) \cdot \lceil \frac{n_B}{2k_{\text{ub}}} \rceil + \left(\sum_{i=0}^{k_{\text{ub}}-1} avg_t(G_i^{\text{out}})\right) \cdot \lfloor \frac{n_B}{2k_{\text{ub}}} \rfloor}{k_{\text{ub}} \cdot \left(\lceil \frac{n_B}{2k_{\text{ub}}} \rceil + \lfloor \frac{n_B}{2k_{\text{ub}}} \rfloor\right)}.$$

Using that

$$\frac{ax + by}{a+b} = x\left(1 - \frac{b}{a+b}\right) + y\left(1 - \frac{a}{a+b}\right)$$

gives that t_{avg} equals

$$\frac{\left(\sum_{i=0}^{k_{\text{ub}}-1} avg_t(G_i^{\text{in}})\right) \cdot \left(1 - \frac{k_{\text{ub}}}{n_B} \lfloor \frac{n_B}{2k_{\text{ub}}} \rfloor\right) + \left(\sum_{i=0}^{k_{\text{ub}}-1} avg_t(G_i^{\text{out}})\right) \cdot \left(1 - \frac{k_{\text{ub}}}{n_B} \lceil \frac{n_B}{2k_{\text{ub}}} \rceil\right)}{k_{\text{ub}}}. \tag{4.19}$$

Because, by assumption, k_{ub} is a divisor of n_B, $\lfloor \frac{n_B}{2k_{\text{ub}}} \rfloor$ and $\lceil \frac{n_B}{2k_{\text{ub}}} \rceil$ both equal $\frac{n_B}{2k_{\text{ub}}}$ or are $\frac{1}{2}$ smaller and larger than $\frac{n_B}{2k_{\text{ub}}}$, respectively. If we define x as 0 in the first case and as $\frac{1}{2}$ in the second, then we can write (4.19) as

$$\frac{\left(\sum_{i=0}^{k_{\text{ub}}-1} avg_t(G_i^{\text{in}})\right) \cdot \left(\frac{1}{2} + x \cdot \frac{k_{\text{ub}}}{n_B}\right) + \left(\sum_{i=0}^{k_{\text{ub}}-1} avg_t(G_i^{\text{out}})\right) \cdot \left(\frac{1}{2} - x \cdot \frac{k_{\text{ub}}}{n_B}\right)}{k_{\text{ub}}}.$$

From the definitions of G_i^{in} and G_i^{out}, $0 \le i < k_{\text{ub}}$, it follows that $avg_t(G_i^{\text{in}}) \ge avg_t(G_i^{\text{out}})$. Hence, removing the term $x \cdot \frac{k_{\text{ub}}}{n_B}$ from the previous expression will not increase its value. Furthermore, it follows from the definitions of G_i G_i^{in}, G_i^{out}, $1 \le i < k_{\text{ub}}$, that $avg_t(G_i^{\text{out}}) \ge avg_t(G_{i-1}^{in})$. Hence, $\sum_{i=0}^{k_{\text{ub}}-2} avg_t(G_i^{\text{out}}) \ge \sum_{i=1}^{k_{\text{ub}}-1} avg_t(G_i^{\text{in}})$. Consequently, t_{avg} is at least

$$\frac{\frac{1}{2} \cdot \left(\sum_{i=0}^{k_{\text{ub}}-1} avg_t(G_i^{\text{in}})\right) + \frac{1}{2} \cdot \left(\sum_{i=1}^{k_{\text{ub}}-1} avg_t(G_i^{\text{in}}) + avg_t(G_{k_{\text{ub}}-1}^{\text{out}})\right)}{k_{\text{ub}}}. \tag{4.20}$$

The term $avg_t(G_{k_{\text{ub}}-1}^{\text{out}})$ is at least $t(n_B - 1)$. Moreover, $t(0) \ge avg_t(G_0^{in})$. As a result, the expression given

by (4.20) plus $\frac{\frac{1}{2}\cdot(t(0)-t(n_B-1))}{k_{ub}}$ is at least

$$\frac{\left(\sum_{i=0}^{k-1} avg_t(G_i^{in})\right)}{k_{ub}}. \tag{4.21}$$

By the definition of G_i, $0 \leq i < k_{ub}$, this expression equals $avg_t(X)$. Hence, (4.20), for which we derived that it is at most t_{avg}, plus $\frac{\frac{1}{2}\cdot(t(0)-t(n_B-1))}{k_{ub}}$ is at least $avg_t(X)$. This yields that (4.18) holds, which we had to prove. □

Theorem 11 assumes that k_{ub} is a divisor of n_B. However, using the same arguments as for Theorem 8, it can be verified from the proof of Theorem 11 that the theorem also approximately holds in the case that k_{ub} is not a divisor of n_B.

As discussed in Section 3.2.2, an assignment constructed by Algorithm 2 is feasible, i.e., (3.9) holds, if and only if the left-hand side of (4.16) is at most $k_{ub} \cdot t_d$. Hence, by Theorem 11, a feasible assignment is guaranteed to be found by Algorithm 2 if $t_d \geq t_{avg} + \frac{\alpha}{k_{ub}} + \frac{t(0)-t(n_B-1)}{2k_{ub}}$. For a given B and n, the function t is constant. Consequently, the expression $\frac{t(0)-t(n_B-1)}{2k_{ub}}$ is descending in k_{ub}. From the definition of α, it follows that α will roughly decrease when k_{ub} increases. Consequently, $\frac{\alpha}{k_{ub}}$ is descending in k_{ub}, as well. Hence, the lower bound on t_d for which it is guaranteed that Algorithm 2 gives a feasible solution approaches t_{avg} when k_{ub} increases.

In Section 3.2.2, we presented Algorithm 1 as an heuristic for constructing the assignment according to which the movie has to be stored on the disk, such that the required buffer size is minimized. This algorithm tries to find a an feasible assignment for an increasing k_{ub} that is at most n_B, where an assignment is feasible if (3.9) holds or, equivalently, if (3.17) does not exceed $k_{ub} \cdot t_d$. The algorithm terminates whenever such an assignment is encountered and subsequently outputs that assignment. The problem of finding a feasible solution for a given k_{ub} has been discussed in this chapter. As discussed above, the upperbound on t_d for which it is guaranteed that a feasible assignment is found approaches t_{avg} for an increasing k_{ub}. Because, by assumption, n_B is large, this implies that for all $t_d > t_{avg}$, Algorithm 1 outputs a feasible assignment. Furthermore, by Theorem 8, a feasible assignment does not exist if $t_d < t_{avg}$. Consequently, if a feasible solution exist for any k_{ub}, our approach also finds one, except possibly if $t_d \approx t_{avg}$. However, the value of k_{ub} does not have to be minimal, which is the result of using a heuristic.

It can easily be verified that Theorem 9 also holds if the equations are strict inequalities. Hence, a video-on-demand system can be defined for a given number of users if and only if the equation $B > c_{max}(n \cdot t_d + s(n))$ can be fulfilled in the case that the buffer requirements are given by Theorem 1 and if and only if $B > c_{max}(n \cdot t_d + s_2(n))$ can be fulfilled in the case that the buffer requirements are given by Theorem 4. Clearly, these equations can be fulfilled if and only if the maximum number of admitted users does not exceed $\frac{r_{avg}}{c_{max}}$.

We still have the freedom in choosing permutation π. With the choice of the permutation we aim at preventing a accumulation of positions with a low transfer rate in the sequence of the assignment, because this is at cost of required buffer size. In the simulation program, for which the results are presented in Chapter 6, we define π such that sets with a large average transfer time are alternated by sets with a small average transfer rate. Formally, we define π as follows.

$$\pi(i) = \begin{cases} \frac{i}{2} & \text{if} \quad i \bmod 2 = 0 \\ k_{ub} - 1 - \frac{i-1}{2} & \text{if} \quad i \bmod 2 = 1. \end{cases} \tag{4.22}$$

Hence, the ordering of the sets G_0 through $G_{k_{ub}-1}$ is $G_0, G_{k_{ub}-1}, G_1, G_{k_{ub}-2}, \ldots$. Note that the definition of π is similar to the definition of a_2.

Example 1 revisited. Figure 4.11 gives the upperbound on the value of t_d, denoted by t_d^{gar}, for which, by Theorem 11, it is guaranteed that the heuristic gives a feasible assignment. Hence, $t_d^{gar} = t_{avg} + \frac{\alpha}{k_{ub}} + \frac{t(0)-t(n_B-1)}{2k_{ub}}$. The figure also depicts t_{avg}. The permutation π is defined by (4.22). Furthermore, it is assumed that each of the six zone of the disk consists of 10000 positions, which implies a disk capacity of 9.8 GByte.

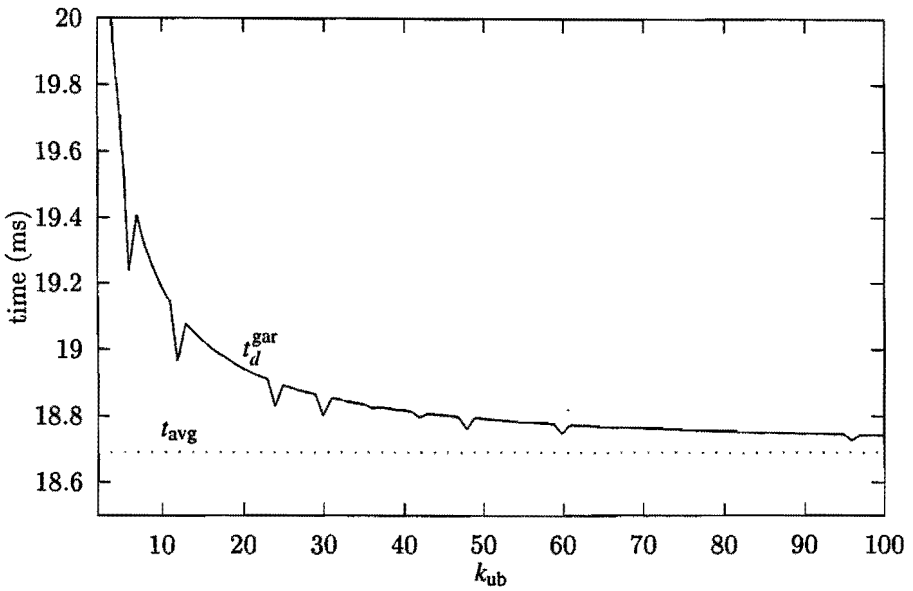

Figure 4.11. Lower bounds on t_d given by Theorem 11

The figure has the following interpretation. If t_d lies below the line that indicates t_{avg}, then a feasible assignment does not exist. If t_d lies between the line indicating t_{avg} and the line indicating t_d^{gar}, then a feasible assignment may be found, but this is not guaranteed and if t_d lies above the line indicating t_d^{gar}, then it is guaranteed that a feasible assignment will be found.

We conclude with a quantification of the figure. The value of t_{max}, which equals 22.27 ms, is 19.2% larger than t_{avg}. From the figure, it follows that for $k_{\text{ub}} = 10$ the value of t_d^{gar} is 2,7% larger t_{avg} and for $k_{\text{ub}} = 50$ the value is 0.5% larger t_{avg}. □

Chapter 5

Response times

In Theorems 1, 3 and 4, sufficient buffer capacities are given for the case that users do not send requests to the server. In Section 5.1, we present an algorithm for handling requests for the case that the buffer requirements are given by Theorem 1. In Sections 5.2 and 5.3, we do the same for the case that the buffer requirements are given by Theorems 3 and 4, respectively.

5.1 Handling requests for revised TB, unconditional solution

In this section, we present an algorithm for handling requests in the case that the buffer size of the users is given by (2.7), such that safeness remains guaranteed. This means that the algorithm has to determine when te first requested data block of a new user may be fetched and when the user may start consuming the requested data.

Before we present the algorithm, we introduce some notation and we make some assumptions. We define the function $d : \{0,1,\ldots,n\} \rightarrow L \cup \{n_B\}$, where $d(u) = i$ indicates that data block i is the next data block that user u wants to be fetched. If user u does not want a data block, i.e. either the user does not want to watch the movie or the last data block of the movie has just been fetched for the user, then $d(i) = n_B$, by definition. The value i_u gives the first data block that has to be fetched for user u when no request arrives at the server, i.e. i_u is raised by one during a sweep in the case that a data block is fetched for u in that sweep and does not change, otherwise. As long as user u does not send a request to the server $d(u) = i_u$ and at the moment that the user sends a request $d(u)$ becomes different from i_u. If a request is sent by user u, then, by assumption, the user stops consuming and the server flushes the corresponding buffer. The user may start consuming the requested data, when a sign is given by the server. If in the meantime a data block other than $d(u)$ is placed in the buffer, because it takes time to respond to a request, this data block is flushed as well.

The time required for handling a request consists of two parts, namely the time it takes before the first requested data block may be fetched and the time it takes from this moment until the moment that the user may start consuming (see Figure 5.1). We first discuss the first part. We say that the request of u is granted, whenever it is allowed that the first requested data block of u is fetched. By assumption, no data block is fetched for a user between the moment that his/her request arrives at the server and the moment that the request is granted, except possibly for the data block that is fetched in the sweep that is executed at the moment the request arrives at the server. As mentioned, this data block is flushed. When a data block is fetched for user u, this is data block i_u, by definition. Consequently, when a request of user u is granted, the value $d(u)$ must have been assigned to i_u. We let the assignment $i_u := d(u)$ indicate that the request of u is granted, by definition.

Next, we discuss the time that passes between the moment a request is granted and the moment that the

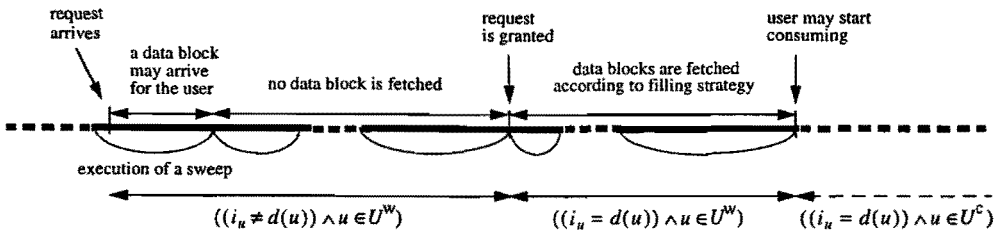

Figure 5.1. The handling of a request depicted on a time axis.

Algorithm 3 Algorithm for handling requests for the case that the buffer capacity is given by (A.1).

forall $1 \leq u \leq n$ /*granting each request*/
  $i_u := d(u)$;
$U^w := \{u | i_u < n_B\}$; /*initialize U^w*/
$U^i := \{u | i_u = n_B\}$; /*initialize U^i*/
$U^c := \emptyset$; /*initialize U^c*/
while true
  $U^w := U^w \cup \{u | i_u \neq d(u)\}$; /*updating U^w*/
  $U^i := U^i \setminus \{u | i_u \neq d(u)\}$; /*updating U^i*/
  $U^c := U^c \setminus \{u | i_u \neq d(u)\}$; /*updating U^c*/
  forall $u \in \{u | d(u) = n_B\}$ /*granting request of each user that wants to stop watching*/
    $i_u := d(u)$;
    $U^i := U^i \cup \{u\}$;
  determine $A \subseteq U^w$ such that /*determine users that may start consuming*/
    $\forall_{u \in A}$ u has at least $1 + \lceil \sum_{u \in U} \frac{\sigma_1(i_u)}{sb_u} + (n - |U|) \cdot \frac{\tau}{sb_u} \rceil$ data blocks in his/her buffer or $i_u = n_B$;
  $U^w := U^w \setminus A$;
  $U^c := U^c \cup A$;
  determine $C \subseteq \{u | i_u \neq d(u)\}$ such that /*determine users whose request is granted*/
    $\forall_{u \in V}$ u has at least $1 + \sum_{u \in U \setminus C} \frac{\sigma_1(i_u)}{sb_u} + \sum_{u \in C} \frac{\sigma_1(d(u))}{sb_u} + (n - |U \cup C|) \cdot \frac{\tau}{sb_u}$ data blocks in his/her buffer;
  forall $u \in C$
    $i_u := d(u)$;
  execute sweep;

corresponding user may start consuming. We divide the set of users into the sets U^w, U^c and U^i. The set U^w contains the users whose request has arrived at the server and and who are waiting for a sign that they can (re)start consuming. The set U^i contains the users that do not want to watch a movie and the set U^c contains the consuming users. The moment that a sign is sent to a user that he/she may start consuming is now equivalent to the moment that u is transferred from U^w to U^c. If a user has consumed the last data block of a movie, then he/she is transferred from U^c to U^i.

Algorithm 3 determines how requests of users can be handled, in the case that the buffer requirements are given by Theorem 1, i.e., it determines for each user u when $d(u)$ is assigned to i_u, which corresponds to granting the request of u, and when u may start consuming or stop watching the movie, which corresponds to transferring u from U^w to U^c and U^i, respectively. In the algorithm V is defined as the set of users that are consuming and for whom the last data block of the movie still has not been fetched, i.e. $V = U^c \cap U$. Hence, the users from V are the only users whose buffer may underflow. Because U^w, U^i and U^c are only used between the execution of two sweeps, the algorithm updates these sets at the end of each sweep instead of during the sweeps. The response time is defined as the time between the arrival of a request at the server and the moment the user may (re)start consuming. Hence, the response time is the time between the arrival of a request and the moment the user is moved from U^w to U^c or from U^w to U^i (see Figure 5.1).

In the initialisation of Algorithm 3 the requests of all users are granted and each user who wants to consume is put into U^w. Hence, for each user who wants a data block, a data block may be fetched, but no user is allowed to consume. Before the start of each sweep the algorithm determines a set of users that may start

consuming and a set of users, for which a request is is granted. First the requests of each user that wants to stop consuming is granted, which, by definition, implies that the user is also transferred from U^w to U^i. Next, the set of users that may start consuming, in the algorithm denoted by A, is determined. The set consists of the users, for whom the last data block of the movie has been fetched and the users that have at least the number of data blocks given by Expression (2.6) in their buffer. For the interpretation of the expression we refer to Section 2.1.1. Note that in the algorithm the ceiling is taken from the expression. The reason is that only complete data blocks are fetched. Finally, a set C is determined which consists of the users whose request is granted. The set is chosen such that after granting the requests, each user from V still has at least the number of data blocks given by (2.6) in the corresponding buffer. The next theorem states that safeness is guaranteed when Algorithm 3 is used.

Theorem 12. *If the buffer capacity is given by (2.7), $t_d \geq t(n_B - 1)$ and filling strategy TB is used for each user whose request has been granted, then safeness is guaranteed whenever the requests of the users are handled according to Algorithm 3.*

Proof. From the definition of filling strategy TB, it follows that no buffer overflow occurs. Buffer underflow can only occur for users from V, as follows from its definition. We prove by induction to the number of executed sweeps, denoted by w, that just before the start of a sweep, i.e., just before the server reaches the last line of Algorithm 3, each user from V has at least the number of data blocks given by (2.6) in the buffer. Because this number is sufficient to survive sweep w, as we showed in the proof of Theorem 1, this proves the theorem.

We start with the case that $w = 0$. Just before the determination of A, $U^c = \emptyset$. Initially, all buffers are empty. As a result, $A = \emptyset$. Hence, the assignment $U^c := U^c \cup A$ does not affect U^c. Consequently, $U^c = \emptyset$ at the start of the first sweep, which implies that $V = \emptyset$. Hence, the induction hypothesis trivially holds for $w = 0$.

We now prove the induction hypothesis for sweep $w = m + 1$. We may assume that at the start of sweep m, the buffer of each user from V contains at least the number of data blocks given by (2.6). As showed in the proof of Theorem 1, each user from V has again at least the number of data blocks given by (2.6) in the buffer at the end of sweep w. Updating U^w, U^i and U^c does not affect this because this only reduces the set V. We now prove that granting a request to stop consuming and putting the corresponding user into U^i does neither affect this. To prove that each user from V has still at least the number of data blocks given by (2.6) in the buffer after granting the request of user u, where $d(u) = n_B$, it suffices to prove that $i_u := n_B$, $1 \leq u \leq n$ does not increase Expression (2.6). From the definition of U, it follows that this corresponds to proving that

$$1 + \sum_{u' \in U} \frac{\sigma_1(i_{u'})}{sb_u} + (n - |U|) \cdot \frac{\tau}{sb_u} \geq 1 + \sum_{u' \in U \setminus \{u\}} \frac{\sigma_1(i_{u'})}{sb_u} + (n - |U \setminus \{u\}|) \cdot \frac{\tau}{sb_u}.$$

If $u \notin U$, then the equation clearly holds. Otherwise, it is implied by $\sigma_1(i_u) \geq \tau$. By the definition of σ_1, $\sigma_1(i_u) \geq t_a(i_u) - t_d$. Since t is descending, $t_a(i) - t_d \geq t(n_B - 1) - t_d = \tau$. Hence, $i_u := n_B$ does not increase Expression (2.6). Furthermore, because $u \notin U$ holds after the assignment $i_u := n_B$, u is not added to V. Hence, each user of V has at least the number of data blocks in the buffer given by (2.6) if each request to stop consuming has been granted and the corresponding users are put into U^i. This is also true after the users from A are put into U^c, as follows from the definition of A.

Because the requests to stop consuming are granted before the determination of C, $d(u) \neq n_B$ for all $u \in C$. Hence, as a result of granting the requests of all users from C, the users from C are added to the set U. Furthermore, because $i_u = d(u)$ for all $u \in U^c$, $C \cap U^c = \emptyset$. Consequently, the set V, which by definition is $U^c \cap U$, does not change after the requests have been granted. From these observations and the definition of C, it follows that after the assignments the buffer of each user from V still contains at least the number of data blocks given by (2.6). This proves the induction hypothesis and with that the theorem. □

The response time depends on how A and C are determined. From the proof of Theorem 12, it follows that the safeness of Algorithm 3 is based on the property that at any moment between two sweeps each

user from V has at least the number of data blocks given by (2.6) in the buffer. Between two arbitrary successive sweeps, we define x_u, $u \in V$, as the number of data blocks that u has more in the buffer than this expression, which is thus at least 0. Consider the moment that C is determined between two arbitrary successive sweeps. Let u be the user from V with the least amount of data in the buffer at this moment. Hence, u is the user for whom x_u is minimal. As stated in Algorithm 3, C is determined such that u, and with that each user from V, has at least

$$1 + \sum_{u \in U \setminus C} \frac{\sigma_1(i_u)}{sb_u} + \sum_{u \in C} \frac{\sigma_1(d(u))}{sb_u} + (n - |U \cup C|) \cdot \frac{\tau}{sb_u}$$

data blocks in the buffer. By the definition of x_u this condition is equivalent to the condition that this number minus Expression (2.6) is a lowerbound on x_u. Hence, the condition is equivalent to

$$x_u \geq \sum_{u \in C} \frac{\sigma_1(d(u))}{sb_u} - \sum_{u \in C \cap U} \frac{\sigma_1(i_u)}{sb_u} - |C \setminus U| \cdot \frac{\tau}{sb_u}. \tag{5.1}$$

Consequently, the value x_u gives a upperbound on how much Expression (2.6) may increase as a result of granting a set of requests.

We define $\alpha(u')$ as the number of complete sweeps that user u' is waiting for his/her request being granted since it arrived at the server. Moreover, we define W as the set of users that are waiting at least one complete sweep. Hence, $W = \{u | \alpha(u) \geq 1\}$. If (5.1) holds for $C = W$, then the request of each user from W is granted, i.e. if

$$\min_{u \in V} x_u \geq \sum_{u \in W} \frac{\sigma_1(d(u))}{sb_u} - \sum_{u \in W \cap U} \frac{\sigma_1(i_u)}{sb_u} - |W \setminus U| \cdot \frac{\tau}{sb_u}. \tag{5.2}$$

By assumption, the minimum over an empty set is ∞. Hence, if $V = \emptyset$, then (5.2) does always hold. If (5.2) does not hold, then we have to choose which requests are granted and which requests are not granted. We thereby aim at a low worst-case response time. We use the following selection criterion. The request of user $u' \in W$ is granted if and only if

$$\min_{u \in V} x_u \cdot \frac{\alpha(u')}{\sum_{u \in W} \alpha(u)} \geq \sigma_1(d(u')) - |i_{u'} < n_B| \cdot \frac{\sigma_1(i_{u'})}{sb_u} - |i_{u'} = n_B| \cdot \frac{\tau}{sb_u}. \tag{5.3}$$

The right-hand side gives the contribution of user u' to the right-hand side of Equation (5.1) if $u' \in C$, i.e. it gives the increase of Expression (2.6) as a result of granting the request of user u'. For each user that wants his/her request to be granted, a part of $\min_{u \in V} x_u$ that is linear in the time the user is waiting is reserved. Hence, the longer a user waits the more the user is privileged when the set of users is determined whose request is granted.

We want to prevent that the choice of the set of users that may start consuming, i.e., the choice of A, affects the choice of C. It is easy to verify that this is the case if for each $u \in A$ it holds that either $i_u = n_B$, which implies that $u \notin V$ when u is added to U^c, or

$$x_u \geq \sum_{u \in W} \frac{\sigma_1(d(u))}{sb_u} - \sum_{u \in W \cap U} \frac{\sigma_1(i_u)}{sb_u} - |W \setminus U| \cdot \frac{\tau}{sb_u} \tag{5.4}$$

or $x_u \geq \min_{u' \in V} x_{u'}$.

We now investigate the worst-case response time of the presented approach. Before we give the lemma that states how much time it maximally takes before a request is granted, we give insight in the correctness of the lemma. By definition, a data block is large enough to survive a sweep in which for each user a data block is fetched, such that the switch time is maximal and the transfer time required for each user is t_d. As discussed in Section 2.1.2 (see Expression (2.4)), the number of data blocks that is minimally saved by fetching only $n - 1$ data blocks instead of n is given by

$$\frac{t_d + s(n) - s(n-1)}{sb_u}. \tag{5.5}$$

When a request of a user arrives at the server, no data block is fetched for the user anymore until the request is granted, except for the data block that is possibly fetched in the sweep during whose execution the request arrives at the server (see Figure 5.1). Consequently, when a user is waiting for, say, m sweeps, at least $(m-1)$ times the number of data blocks given by (5.5) is saved by the user. As proved in Theorem 12 $\sigma_1(i) \geq \tau$, for all $i \in L$. Consequently, the increase of Expression (2.6) resulting from granting the request of an arbitrary user, i.e. the right hand side of Expression (5.3), is maximally $\frac{\max_{0\leq i<n_B}\sigma_1(i)}{sb_u} - \frac{\tau}{sb_u}$. Hence, if a user waits $1+\gamma$ sweeps, where

$$\gamma = \left\lceil \frac{\max_{0\leq i<n_B}\sigma_1(i) - \tau}{t_d + s(n) - s(n-1)} \right\rceil, \tag{5.6}$$

then the number of data blocks that is saved during this sweeps is sufficient to compensate the increase of the required degree of filling of the buffers of the consuming users resulting from granting his/her request, i.e. the increase in Expression (2.6). The following lemma states that it takes indeed at most $1+\gamma$ sweeps before a request is granted.

Lemma 5. *Let $t(n_B-1) \leq t_d$ and $t(n_B-1) < t(0)$. Then it takes at most $1+\gamma$ sweeps before a request is granted, where γ is defined by (5.6).*

Proof. We first prove that $\gamma \geq 1$. By the definition of σ_1, $\max_{0\leq i<n_B}\sigma_1(i) \geq t(0) - t_d$. This and the definition of τ yields that the numerator in the definition of γ is at least $t(0) - t(n_B-1)$. Because, by assumption, $t(n_B-1) < t(0)$ and because $t_d > 0$, this implies that $\gamma \geq 1$.

We prove by induction to the number of executed sweeps, denoted by w, that just before C is determined either

$$\min_{u\in V} x_u \geq \sum_{u\in W} \alpha(u) \cdot \frac{t_d + s(n) - s(n-1)}{sb_u} \tag{5.7}$$

holds or (5.2) holds. First we show that this suffices to prove the lemma.

Let u be a user, who sends a request to the server. It takes at most one sweep before the next sweeps starts. From Algorithm 3, it follows that the request is granted immediately if $d(u) = n_B$. Hence, because $\gamma \geq 1$, the lemma holds in the case that $d(u) = n_B$.

Assume that $d(u) < n_B$ and assume that the request is not granted after $1+\gamma$ sweeps. This means that there is a sweep w, such that between sweep w and $w+1$, $\alpha(u) = \gamma$ and $u \notin C$. We prove that this leads to a contradiction. If (5.2) holds, just before C is determined, then each request from W is granted. Because $\alpha(u) = \gamma$ and $\gamma \geq 1$, user u is in W. Consequently, the request of u is granted, which gives the contradiction. Assume that (5.7) holds. This can be rewritten to

$$\frac{\min_{u\in V} x_u}{\sum_{u\in W}\alpha(u)} \geq \frac{t_d + s(n) - s(n-1)}{sb_u}.$$

As a result, (5.3) is implied by

$$\frac{t_d + s(n) - s(n-1)}{sb_u} \cdot \gamma \geq \frac{\sigma_1(d(u))}{sb_u} - |i_u < n_B| \cdot \frac{\sigma_1(i_u)}{sb_u} - |i_u = n_B| \cdot \frac{\tau}{sb_u}. \tag{5.8}$$

In the proof of Theorem 12 we showed that $\sigma_1(i_u) \geq \tau$. Consequently, the right-hand side is at most $\max_{0\leq i<n_B}(\sigma_1(i)/sb_u) - \tau/sb_u$. This and the definition of γ yields that (5.8) holds. Consequently, the request of u is granted, which gives the contradiction.

Next, we prove by induction to the number of executed sweeps that either (5.2) or (5.7) holds just before C is determined. If $w = 0$ then $V = \emptyset$. Hence, (5.7) holds trivially. Consider the case $w = m+1$, $m \geq 0$. If $V = A$ just before C is determined, then it follows from the definition of A that (5.2) holds. The same holds if (5.2) holds for all $u \in V \setminus A$. Assume that both cases do not hold. By the definition of A, this means that there exist a user u from $V \setminus A$ for which x_u is minimal, i.e., $x_u = \min_{u'\in V} x_{u'}$. We have to prove that either (5.2) or (5.7) holds, where the left-hand sides of the equations are replaced by x_u. We make a

distinction between whether or not a data block has been fetched for u in the last sweep, i.e. in the $(m+1)$th sweep.

Assume that no data block has been fetched for u in the $(m+1)$th sweep. In the proof of Theorem 1 we showed that the buffer of u contains at least the number of data blocks given by (A.8) at the end of the sweep. Updating $i_{u'}$ and U yields that the buffer of u contains at least

$$1+\sum_{u'\in D\cap U}\frac{\sigma_1(i_{u'})}{sb_u}+\sum_{u'\in D\setminus U}\frac{t_a(i_{u'}-2)-t_d}{sb_u}+(n-|D|)\cdot\max_{0\le i<n_B}\frac{\sigma_1(i)}{sb_u},$$

data blocks, where D is the set of users, for whom a data block has been fetched in the $(m+1)$th sweep. Subtracting Expression (2.6) gives an lowerbound for x_u. Hence,

$$x_u\ge(n-|D|)\cdot\max_{0\le i<n_B}\frac{\sigma_1(i)}{sb_u}-\sum_{u\in U\setminus D}\frac{\sigma_1(i_u)}{sb_u}-(n-|U\cup D|)\cdot\frac{\tau}{sb_u},$$

where we use that $\tau\ge t_a(i_{u'}-2)-t_d$ for all $u'\in D\setminus U$. Because $\alpha(u')\ge 1$ for each $u'\in W$, $W\cap D=\emptyset$. As a result, x_u is at least

$$|W|\cdot\max_{0\le i<n_B}\frac{\sigma_1(i)}{sb_u}-\sum_{u\in W\cap U}\frac{\sigma_1(i_u)}{sb_u}-|W\setminus U|\cdot\frac{\tau}{sb_u}. \tag{5.9}$$

As showed in the proof of Theorem 12, granting the requests of users that want to stop consuming does not increase the value of Expression (2.6). Hence, x_u is still at least Expression (5.9), where W, i_u and U reflect the situation before the requests are granted. Granting the request of, say, user u' only affects the expression if $u'\in W$. Assume that this is the case. From the definition of W, it follows that if the the request of u' is granted, u' is removed from W. Hence, (5.9) increases by

$$-\max_{0\le i<n_B}\frac{\sigma_1(i)}{sb_u}+|i_{u'}^{\text{old}}<n_B|\cdot\frac{\sigma_1(i_{u'}^{\text{old}})}{sb_u}+|i_{u'}^{\text{old}}=n_B|\cdot\frac{\tau}{sb_u}.$$

In the proof of Theorem 12 we also showed that $\sigma_1(i)\ge\tau$, for all $0\le i<n_B$. Hence, the expression is not positive. As a result, (5.9) is also a lowerbound for x_u if the values occurring in it are updated. Consequently, (5.2) holds just before C is determined and $m+1$ sweeps have been executed.

Assume that a data block is fetched for u in the $(m+1)$th sweep. By definition, $u\in V\setminus A$. Because a user can only be added to V via A, u is also an element of V just before C is determined and w sweeps have been executed. Consider this moment. Because $u\in V$, x_u is defined. We first prove that just before the $(m+1)$th sweep is executed (5.7) holds. By the induction hypothesis either (5.2) or (5.7) holds. If (5.2) holds, then each request from W is granted. Hence, $W=\emptyset$ after the requests have been granted. Because $x_u\ge 0$ is invariant, as we showed in Theorem 12, (5.7) holds at the start of the $(m+1)$th sweep. Assume that (5.7) holds and (5.2) does not hold just before C is determined. As a result of granting the requests from C, x_u decreases exactly as much as Expression (2.6) increases, as follows from the definition of x_u. Hence, x_u decreases by

$$\sum_{u\in C}\frac{\sigma_1(d(u))}{sb_u}-\sum_{u\in C\cap U}\frac{\sigma_1(i_u)}{sb_u}-|C\setminus U|\cdot\frac{\tau}{sb_u}$$

data blocks. Because (5.2) does not hold, selection criterion (5.3) is applied. Consequently, the expression is at most

$$x_u\cdot\frac{\sum_{u\in C}\alpha(u)}{\sum_{u\in W}\alpha(u)}$$

Hence, after the requests from the users from C have been granted, x_u is at least x_u minus this expression. Because (5.7) holds, $x_u\ge\sum_{u\in W}\alpha(u)\cdot\frac{t_d+s(n)-s(n-1)}{sb_u}$ just before C is determined. Hence, after the requests

have been granted x_u is at least

$$\left(1 - \frac{\sum_{u \in C} \alpha(u)}{\sum_{u \in W^{\text{old}}} \alpha(u)}\right) \cdot \sum_{u \in W^{\text{old}}} \alpha(u) \cdot \frac{t_d + s(n) - s(n-1)}{sb_u},$$

where W^{old} is the set W before the requests are granted. This can be written to

$$\sum_{u \in W^{\text{old}} \setminus C} \alpha(u) \cdot \frac{t_d + s(n) - s(n-1)}{sb_u}.$$

Because granting the requests results in the assignment $W := W \setminus C$, this implies that (5.7) holds at the start of sweep $w+1$. As a result, both if (5.2) holds and if (5.7) holds, then the buffer of u contains at least the number of data blocks given by (5.7) just before the $(m+1)$th sweep is executed. We now prove that this is also true after the execution of the sweep.

In the proof of Theorem 1 if the buffer of a user contains the number of data blocks given by (2.6) at the start of the sweep and if a data block is fetched for the user, then the buffer contains at least the number of data blocks given by (A.4) at the end of the sweep. In terms of x_u, $u \in V$, this means that if $x_u = 0$ at the start of the sweep, then x_u is at least $\frac{n-|D| \cdot t_d + s(n) - s(|D|)}{sb_u}$ at the end of the sweep. Similarly, it can be proved that for an arbitrary starting value of x_u, x_u is raised by this fraction. By definition, $u' \in W$ implies that $\alpha(u') \geq 1$. Hence, in the $(m+1)$th sweep no data block is fetched for a user from W, where W is based on the situation just before C is determined and $(m+1)$ sweeps have been executed, i.e. $D \cap W = \emptyset$. As a result, x_u is raised by at least

$$|W| \cdot \frac{t_d + \min_{1 \leq i \leq n}(s(i) - s(i-1))}{sb_u}. \tag{5.10}$$

It can be shown that $\min_{1 \leq i \leq n}(s(i) - s(i-1)) = s(n) - s(n-1)$. Hence, Expression (5.10) gives exactly the increase of the right-hand side of Equation (5.7). Consequently, (5.7) holds at the end of the $(w+1)$th sweep. As mentioned above, granting the requests of users that want to stop consuming, does not decrease x_u. Furthermore, no users are added to W, because W only contains users whose request has not been granted. Consequently, the right-hand side of (5.7) does not increase. Hence, (5.7) holds just before C is determined and $w+1$ sweeps have been executed. □

From the definition of x_u, it follows that whenever a user has at least $1 + n \cdot \max_{0 \leq i < n_B} \sigma_1(i)/sb_u$ data blocks in the buffer, (5.4) holds, which means that the user may start consuming. Consequently, the time between the moment that a request is granted and the moment that the user may start consuming is at most the time that is maximally required for fetching $1 + n \cdot \max_{0 \leq i < n_B} \sigma_1(i)$ data blocks. By the used filling strategy, this takes $1 + \eta$ sweeps, where

$$\eta = \lceil n \cdot \max_{0 \leq i < n_B} \frac{\sigma_1(i)}{sb_u} \rceil. \tag{5.11}$$

Lemma 5 and this observation are the key of the following theorem.

Theorem 13. *Let $t(n_B - 1) \leq t_d$ and $t(n_B - 1) < t(0)$. Then the worst-case response time is bounded by the time that is maximally required for $2 + \gamma + \eta$ successive sweeps, where γ and η are defined according to (5.6) and (5.11), respectively.*

Proof. Assume that user u sends a request to the server. By Lemma 5, it takes at most $1 + \gamma$ sweeps before the request is granted. Hence, we have to prove that it takes at most $1 + \eta$ sweeps before u can start consuming after the request has been granted. By definition, the request is granted between the execution of two sweeps, say w and $w+1$. We have to prove that user u can start consuming the requested data at the latest at the start of sweep $w + \eta + 2$. Assume that this is not the case. We show that this leads to a contradiction. If a data block is fetched for u in each sweep from $w+1$ through $w + \eta + 1$, then u has $1 + \eta$ data blocks in the corresponding buffer. By the definition of x_u, this means that (5.4) holds at the moment that A is determined. As a result, $u \in A$, which means that u can start consuming. This gives the

contradiction. What remains is the proof that u fetches a data blocks in each sweep from $w+1$ through $w+\eta+1$. Because u may start consuming if the last data block of the movie has been fetched, the last data block is not fetched in any of these sweeps. From the definition of filling strategy TB, it now follows that, because the buffer of u is empty at the start of sweep $w+1$, a data block is fetched in each sweep from $w+1$ through $w+\eta+1$ if and only if the buffer of u is large enough to store $1+\eta$ data blocks. Because $\lceil x \rceil \leq x+1$, for an arbitrary real value x, $\eta \leq 1+n \cdot \max_{0\leq i<n_B} \sigma_1(i)/sb_u$. Consequently, the buffer of u is large enough if

$$1+(1+n\cdot \max_{0\leq i<n_B} \frac{\sigma_1(i)}{sb_u}) \leq 3+n\cdot \max_{0\leq i<n_B} \frac{\sigma_2(i)}{sb_u} - \delta.$$

Using the definitions of δ and σ_m and multiplying both sides with sb_u, gives that the equation is equivalent to

$$(n-1)\cdot \max_{0\leq i<n_B} \sigma_1(i) \leq sb_u + (n-1)\cdot \max_{0\leq i<n_B} \sigma_2(i) - t_d + s(n) - s(n-1). \tag{5.12}$$

By the definition of sb_u, $sb_u - (t_d + s(n) - s(n-1))$ equals $(n-1)\cdot t_d + s(n-1)$. Hence, if $\max_{0\leq i<n_B} \sigma_1(i) = \max_{0\leq i<n_B} \sigma_2(i)$ or $n=1$ then the equation is clearly true. Assume that the equation does not hold. Then, by the definition of σ_m, $\max_{0\leq i<n_B} \sigma_1(i) = t(0) - t_d$. Furthermore, $\max_{0\leq i<n_B} \sigma_2(i) \geq t(0) + t(n_B-1) - 2t_d$. Using this and dividing both sides by $n-1$ yield that (5.12) is implied by

$$t(0) - t_d \leq t_d + t(0) + t(n_B-1) - 2t_d,$$

which trivially holds. □

If we want to express the worst-case response time in time units we have to determine the time that is maximally required for a number of successive sweeps. The following theorem states that in the worst-case situation, each sweep of an arbitrary set of successive sweeps can be worst-case, i.e., in each sweep a data block can be fetched for all users at a minimum transfer rate, such that the switch time is maximal.

Theorem 14. *The worst-case time required for m successive sweeps is* $m\cdot(n\cdot t(0)+s(n))$.

Proof. If during a given sweep w each user requests the data block that is stored on position 0, then each user is put into U^w at the end of the sweep. Consequently, V becomes $\emptyset$, which implies that all requests are granted. As a result, the time required sweep $w+1$ can be $n\cdot t(0)+s(n)$. If during this sweep, each user requests again the data block that is stored on position 0, the situation at the start of sweep $w+2$ is similar to the situation at the start of sweep $w+1$. Hence, the time required for sweep $w+1$ is, again, $n\cdot t(0)+s(n)$. If this scenario is repeated m times, we get m successive sweeps with a total execution time of $m\cdot(n\cdot t(0)+s(n))$ time units. Hence, this expression is a lowerbound on the time required for m successive sweeps. Clearly, it is also an upperbound, which proves the theorem. □

We give an example of the meaning of the discussed theorems.

Example 1 revisited. As stated by Theorem 13, the worst-case response time is bounded by the time that is in the worst-case situation required for $2+\gamma+\eta$ successive sweeps. As mentioned, $\max_{0\leq i<n_B} \sigma_1(i) = x$. Consequently, the value of γ is given by $\lceil \frac{x+t_d-t(n_B-1)}{t_d} \rceil$, which equals $\lceil 0.1525 \rceil = 1$. Furthermore, the value of η is $\lceil 0.2096 \rceil = 1$. As a result, the worst-case response time is bounded by the time time that is maximally required for four successive sweeps. By Theorem 14, this time is maximally $4\cdot(n\cdot t(0)+s(n))$, which is 1.51 seconds. For TB the worst-case response time is the time that is maximally required for two successive sweeps, which is $2\cdot(n\cdot t_{min}+s(n))$, where $t_{min} = \frac{280.19}{r_{min}}$. This yields that the worst-case response time is 1.09 seconds.

Consequently, opposite to a saving in the buffer requirements of 36%, we have an increase of the worst-case response time by 39%.

Algorithm 4 Algorithm for handling requests for the case that the buffer capacity is $3 - \delta_2$.

forall $1 \leq u \leq n$
  $i_u := d(u)$;
$U^w := \{u | i_u < n_B\}$;
$U^i := \{u | i_u = n_B\}$;
$U^c := \emptyset$;
while true
  $U^w := U^w \cup \{u | i_u \neq d(u)\}$;
  $U^i := U^i \setminus \{u | i_u \neq d(u)\}$;
  $U^c := U^c \setminus \{u | i_u \neq d(u)\}$;
  forall $u \in \{u | d(u) = n_B\}$
    $i_u := d(u)$;
    $U^i := U^i \cup \{u\}$;
  $A := \{u \in U^w | i_u = n_B \vee$ "u has two data blocks in his/her buffer"$\}$
  $U^w := U^w \setminus A$;
  $U^c := U^c \cup A$;
  $C_1 := \{u \in U^w | i_u \neq d(u) \wedge$ "in the previous two sweeps no data block has been fetched for u"$\}$
  $C_2 := \{u \in U^w | i_u \neq d(u) \wedge$ "in the previous sweep no data block has been fetched for u"
    $\wedge\, t_a(last_u) + t_a(d(u)) \leq 2t_d\}$
  forall $u \in C_1 \cup C_2$
    $i_u := d(u)$;
  execute sweep;

5.2 Handling requests for revised TB, conditional solution

In this section we present an algorithm for handling requests for the case that the buffer requirements are given by Theorem 3. We use the same notation and assumptions as in the previous section. The pseudo code of the algorithm given by Algorithm 4. Because of its similarity with Algorithm 3, the comments are omitted in the algorithm.

As in Algorithm 3, the initialisation consists of granting the requests of all users. Before the start of a sweep, the request of each user who wants to stop consuming is granted. The set A, i.e. the set of users that may start consuming, consists of the users that either have two data blocks in their buffer or have received the last data block of the movie. A request of, say, user u can be granted on the basis of two criteria. Either no data block has been fetched for u for at least two sweeps or no data block has been fetched for one sweep, but $t_a(last_u) + t_a(d(u)) \leq 2t_d$, where $last_u$ is the most recently fetched data block of user u. The equation is to be interpreted as follows. The average transfer time of the most recently transferred data block and the desired next data block does not exceed t_d. If still no data block has been fetched for user u, then $last_u$ is the data block that is stored on position $n_B - 1$, by definition. Since t is descending and because (2.10) holds, this implies that $t_a(last_u) + t_a(d(u)) \leq 2t_d$ regardless of the value of $d(u)$ in that case.

Theorem 15. *If Equation (2.10) holds, the capacity of the buffers is at least* $3 - \delta_2$, *where* δ_2 *is given by (2.12), filling strategy TB is used for each user whose request has been granted and the requests of the users are handled according to Algorithm 4, then safeness is guaranteed.*

Proof. From the proof of Theorem 3, it follows that safeness is guaranteed, if a user from U does not start consuming before he/she has room for strictly less than one data block, if (2.13) remains valid and if in three successive sweeps at most two data blocks are fetched for the same user. We start with proving the first condition. By the definition of A and the assumed buffer size, which is at least $3 - \delta_2$ data blocks, the first requirements is satisfied if $\delta_2 > 0$. By the definition of δ_2, $\delta_2 > 0$ is equivalent to $2 \cdot t_d > t(0) - s_3(2n) + s_2(2n-1)$. Because $s_3(2n) > s_2(2n-1)$, this is implied by $t_d \geq \frac{t(0)}{2}$. This follows from (2.10).

From the definition of C_1 and C_2, it follows that two data blocks that are not successive are never fetched for the same user in two successive sweeps. As a result, (2.13) remains valid, which is our second proof obligation. Finally we prove by contradiction that in three successive sweeps at most two data blocks are fetched for the same user. Let w be the first sweep, such that in the sweeps w, $w+1$ and $w+2$ three data

blocks are fetched for the same user, say u. As mentioned, these data blocks have to be successive. Because a data block is only fetched for a user if $i_u = d(u)$ at the start of the sweep this implies that from the start of sweep w until the start of sweep $w+2$ no request from u arrives at the server.

Consider the start of sweep $w+2$. If $u \notin U^{\mathrm{i}}$, then no data block is fetched for u in sweep $w+2$ which contradicts our assumption that a data block is fetched in sweeps w, $w+1$ and $w+2$.

Assume that $u \notin U^{\mathrm{c}}$ at the start of sweep $w+2$, i.e., u is not a consuming user at the start of sweep $w+1$. As a result, $u \notin A$, which, by definition, means that u has less than two data blocks in the buffer and that $i_u < n_B$. Because the server receives no request from u from the start of sweep w until the start of sweep $w+2$, the buffer is not flushed during this period and $u \notin U^{\mathrm{c}}$ also holds at the start of sweep w and $w+1$. By assumption, a data block is fetched for u in sweep w and $w+1$. Hence, u has at least two data blocks in the buffer at the start of sweep $w+2$, which gives the contradiction.

Assume that $u \in U^{\mathrm{c}}$ at the start of sweep $w+2$. Because, by assumption, a data block is fetched in sweep $w+2$, $i_u < n_B$ at the start of this sweep. We define v as the sweep, at the start of which u starts consuming for the last time, i.e. at the start of which u is moved from U^{w} to U^{c} for the last time. As proved above, the buffer of u has room for strictly less than one data block at the start of sweep v. Consequently, in sweep v no data block is fetched for user u, which implies that $w > v$. Hence, $u \in U^{\mathrm{c}}$ at the start of sweep $w-1$. Because w is chosen minimal, no data block is fetched for user u in $w-1$. As a result, at the start of sweep $w-1$ the buffer of u does not have room for a data block. Another consequence of the minimality of w is that the sweeps $w-1$, w and $w+1$ contain at most two data blocks for the same user. If for a user, say u', two data blocks are fetched in these sweeps that are not successive, then the request to fetch the second block must have been granted between the start of the sweep in which the first data block is fetched and the start of the sweep in which the second is fetched. By the definitions C_1 and C_2 this can only be the case if the first data block is fetched in sweep $w-1$ and the second in sweep $w+1$ and if $t_a(last_{u'}) + t_a(d(u)) \leq 2t_d$. Consequently, the total transfer time required for the two data blocks for u' is bounded by $2t_d$. Furthermore, by (2.10), the transfer time required for a user u'' for who either a single data block or two successive data blocks is fetched during sweeps $w-1$, w and $w+1$ is bounded by $2t_d$, as well. As a result, the time required for these sweeps is bounded by $2n \cdot t_d + s_3(2n)$. Similarly as for the case that $w = m+1$ in the proof of Lemma 1, this leads to a contradiction. □

The next theorem states the worst-case response time of Algorithm 4.

Theorem 16. *In the case that Algorithm 4 is used, the worst-case response time is the time that is maximally required for five successive sweeps, which is at most* $4n \cdot t_d + s_5(4n)$.

Proof. Assume that user u sends a request to the server to jump to data block $d(u)$. We are interested in the maximum time it takes before u can start consuming. If $d(u) = n_B$, u does not want to consume. Hence, we assume that $d(u) < n_B$. It takes at most one sweep from the arrival of the request of u to the start of a sweep. By assumption, no data block is fetched for u, before the request is granted. By the definition of C_1, this takes at most two sweeps. Two sweeps after that, u has two data blocks in the buffer, unless the buffer is not large enough to store two data blocks. Therefore, we have to prove that $3 - \delta_2 \geq 2$, i.e. $\delta_2 \leq 1$. Using the definition of δ_2 gives that $\delta_2 \leq 1$ is equivalent to $2t_d - t(0) + s_3(2n) - s_2(2n-1) \leq n \cdot t_d + \frac{1}{2}s_3(2n)$. Because $t_d \leq t(0)$, which follows from (2.10), and because $n \geq 1$, this is implied by $s_3(2n) \leq 2s_2(2n-1)$. Using (1.1), yields that $2s(2n-1) = s_4(4n-2)$. Again, $n \geq 1$, gives that $s_4(4n-2)$ is at least $s_4(2n)$, which is at least $s_3(2n)$. Hence, $\delta_2 \leq 1$.

By the definition of A, u may start consuming if he/she has two data blocks in the buffer at the start of a sweep. As a result, the worst-case response time is 5 sweeps. We now prove that five sweeps take at most $4n \cdot t_d + s_5(4n)$ time units. We showed in the proof of Theorem 15 that at most two data blocks are fetched for the same user in three successive sweeps. Hence, in five sweeps at most four data blocks are fetched per user. The data blocks fetched for a user are successive, unless he/she sends a request during the sweeps, which is also granted. If the data blocks are successive the maximum transfer time spent for the user is $4t_d$, as follows from the assumption that (2.10) holds. If the data blocks are not successive, but each request is granted via C_2, then the maximum transfer time does not exceed $4t_d$, as well. If two fetched data blocks

are not successive because of a request that is granted via C_1, then, by the definition of C_1, at most three data blocks are fetched for the user. Furthermore, the number of fetched data blocks can only be three if two of them are successive, as follows from the definition of C_1 and C_2. Since t is descending and because of (2.10), we get that the maximum transfer time required for the user is bounded by $t(0)+2t_d$. From (2.10) it also follows that $t(0) \leq 2t_d$. Hence, the maximum transfer time is bounded by $4t_d$. The maximum time required for five successive sweeps is given by the sum of the maximum transfer times spent for each user plus the maximum switch time. Hence, the time that is maximal required for five sweeps is $4n \cdot t_d + s_5(4n)$. □

From the proof of Theorem 16, it follows that in the worst-case scenario a request arrives just after the start of a sweep in which a data block is fetched for the user who sends the request. Hence, the user is consuming and want to make a jump in the movie. If no data block is fetched for a user in at least the last two sweeps at the moment he/she sends the request, because, for example, the user was not watching the movie, then the request is granted at the start of the first sweep after the sweep in which the request arrives, as follows from the definition of C_1. Hence, in this case the request is granted two sweeps earlier than in the worst-case scenario. As a result, the worst-case response time in the case that no data block is fetched for the user that sends the request for at least two sweeps, is three instead of five sweeps. Similarly as for five sweeps, it can be showed that the time that is maximally required for these three sweeps is bounded by $2n \cdot t_d + s_3(2n)$.

5.3 Handling requests for revised dual sweep

Algorithm 5 gives the pseudo code for handling requests in the case that the buffer requirements are given by Theorem 4. The algorithm is similar to Algorithm 3, except for the determination of A and C, which give the set of users that may start consuming and whose requests are granted, respectively. The determination of both sets is based on Expression (2.20) and Expression (2.21), for which we gave the interpretation in Section 2.2. In Algorithm 5, X_w is defined as the set of users for whom no data block has been fetched in sweep w, where the sweeps are numbered from 1 onwards. Furthermore, for any moment between sweep $w-1$ and sweep w, U_w is defined as

$$\{u | i_u \in U \wedge u \in X_{w-1} \wedge i_u = d(u) \wedge \text{"}u \text{ has room for a data block in the buffer"}\}$$

if $w > 1$. If $w = 1$ then the second conjunct in the definition is removed. The set U_w can be interpreted as the set of users for whom a data block is fetched in sweep w if the situation does not change anymore from the moment under consideration until the start of sweep w. This definition corresponds to the meaning of U_w in the proof of Theorem 4. Furthermore, U_w is only meaningful in combination with a moment in time between sweep $w-1$ and sweep w. The following theorem states that safeness is guaranteed when Algorithm 5 is used.

Theorem 17. *If the buffer capacity is given by (2.22), $n \geq 2$, $t(0) \geq t_d \geq t(n_B - 1)$, filling strategy DS is used for each user whose request has been granted and the requests of the users are handled according to Algorithm 5 then safeness is guaranteed.*

Proof. As in Theorem 12, we only have to prove that no buffer underflow occurs for any user from V. We define the predicate $P(w)$ as the property that for all $u \in V$ it holds that user u has has at least the number of data blocks given by (2.21) in the buffer if $u \notin U'_w$ and at least the number given by (2.20), otherwise. We prove by induction to w that $P(w)$ holds at the start of sweep w. Because no buffer underflow occurs in sweep w if $P(w)$ holds at the start of it, as we showed in the proof of Theorem 4, this proves the theorem.

As basis we take the case that $w = 1$. Similarly as in the basis case of the proof of Theorem 12, it can be showed that $V = \emptyset$ at the start of the first sweep. Hence, $P(1)$ holds at the start of the first sweep.

Consider the start of sweep $m+1$, $m \geq 1$. By the induction hypothesis, $P(m)$ holds at the start of sweep m. We showed in the proof of Theorem 4 that at the end of the sweep $P(m+1)$ holds. Updating U^w, U^i and U^c does not affect this because this only reduces the set V. Let u be a user who want to stop watching

Algorithm 5 Algorithm for handling requests for the case that the buffer capacity is given by (A.9).

forall $1 \leq u \leq n$
  $i_u := d(u)$;
$U^{\mathrm{w}} := \{u | i_u < n_B\}$;
$U^{\mathrm{i}} := \{u | i_u = n_B\}$;
$U^{\mathrm{c}} := \emptyset$;
while true
  $U^{\mathrm{w}} := U^{\mathrm{w}} \cup \{u | i_u \neq d(u)\}$;
  $U^{\mathrm{i}} := U^{\mathrm{i}} \setminus \{u | i_u \neq d(u)\}$;
  $U^{\mathrm{c}} := U^{\mathrm{c}} \setminus \{u | i_u \neq d(u)\}$;
  forall $u \in \{u | d(u) = n_B\}$
    $i_u := d(u)$;
    $U^{\mathrm{i}} := U^{\mathrm{i}} \cup \{u\}$;
  determine $A \subseteq U^{\mathrm{w}}$ such that $\forall_{u \in A}$ either u has at least
    $1 + \left\lceil \sum_{u \in U} \frac{\max(\sigma_1(i_u),0)}{sb_{\mathrm{d}}} \right\rceil$ data blocks in his/her buffer if $u \in U_w$ and at least
    $\left\lceil \frac{|U_w| \cdot t_d + s(|U_w|)}{sb_{\mathrm{d}}} + \sum_{u \in U} \frac{\max(\sigma_1(i_u),0)}{sb_{\mathrm{d}}} \right\rceil$ if $u \notin U_w$
    or $i_u = n_B$;
  $U^{\mathrm{w}} := U^{\mathrm{w}} \setminus A$;
  $U^{\mathrm{c}} := U^{\mathrm{c}} \cup A$;
  determine $C \subseteq \{u | i_u \neq d(u)\}$ such that $\forall_{u \in V}$ u has at least
    $1 + \sum_{u \in U \setminus C} \frac{\max(\sigma_1(i_u),0)}{sb_{\mathrm{d}}} + \sum_{u \in C} \frac{\max(\sigma_1(d(u)),0)}{sb_{\mathrm{d}}}$ data blocks in the buffer if $u \notin U_w$ and at least
    $\frac{|U_w \cup (C \cap X)| \cdot t_d + s(|U_w \cup (C \cap X)|)}{sb_{\mathrm{d}}} + \sum_{u \in U \setminus C} \frac{\max(\sigma_1(i_u),0)}{sb_{\mathrm{d}}} + \sum_{u \in C} \frac{\max(\sigma_1(d(u)),0)}{sb_{\mathrm{d}}}$ if $u \notin U_w$;
  forall $u \in C$
    $i_u := d(u)$;
  execute sweep;

the movie. After the request is granted, i.e. after $i_u := n_B$, and after u is put into U^{i}, $\sigma_1(i_u)$ is $-\infty$, $u \notin V$ and u is removed from U_{m+1}. As a result, both (2.21) and (2.20) do not increase and u is not added to V. Consequently, $P(m+1)$ still holds after granting the request of each user who wants to stop watching the movie and putting the user into U^{i}. By the definition of A and because U_{m+1} is not affected by letting users start consuming, $P(m+1)$ also holds after the users of A are put into U^{c}.

Because $C \subseteq U^{\mathrm{w}}$, granting the requests of the users from C does not affect V. The requests to stop watching the movie have already been granted when C is determined. Consequently, $d(u) < n_B$ for all $u \in C$. Hence, as a result of granting the requests of all users from C, U becomes $U \cup C$. Furthermore, because, by assumption, the buffer of each user whose request has not been granted is empty, U_{m+1} becomes $U_{m+1} \cup (C \cap X_m)$ after granting the requests. From these observations and the definition of C, it follows that $P(m+1)$ also holds after the requests of C are granted. Hence, the induction hypothesis holds for $m+1$. □

For the determination of A and C we use the same idea as for Algorithm 3. Consider a moment between the end of sweep $w-1$ and the start of sweep w. The safeness of Theorem 17 is based on the property that at any moment between two successive sweeps the buffer of each user from V contains at least the number of data blocks given by (2.21) if $u \notin U_w$ and at least the number given by (2.20), otherwise. We define y_u, $u \in V$, as the number of data blocks in the buffer of u minus Expression (2.21) if $u \notin U_w$ and minus Expression (2.20), with U_w replaced by X_{w-1}, otherwise. Note that by the definition of C and because U_w can be a strict subset of X, unlike x_u, y_u can be negative. Similarly as the condition that C has to answer for each $u \in V$ in Algorithm 3 can be written as condition (5.1), the condition that C has to answer for each $u \in V \cap U_w$ in Algorithm 5 is equivalent to

$$y_u \geq \sum_{v \in C} \frac{\max(\sigma_1(d(v)),0)}{sb_{\mathrm{d}}} - \sum_{v \in C \cap U} \frac{\max(\sigma_1(i_v),0)}{sb_{\mathrm{d}}}. \tag{5.13}$$

Because, by definition, $U_w \subseteq X_{w-1}$, we have that

$$\frac{|U_w \cup (C \cap X_{w-1})| \cdot t_d + s(|U_w \cup (C \cap X_{w-1})|)}{sb_{\mathrm{d}}} \leq \frac{|X_{w-1}| \cdot t_d + s(|X_{w-1}|)}{sb_{\mathrm{d}}}.$$

By this and the definition of y_u, the condition that C has to answer for each $u \in V \setminus U_w$ is implied by (5.13) as well.

We define $\alpha_2(u)$, $1 \leq u \leq n$, as $\frac{\alpha(u)-1}{2}$, where $\alpha(u)$ is, again, defined as the number of complete sweeps that u is waiting for his/her request being granted since it arrived at the server. Furthermore, we redefine W as $W = \{u | \lfloor \alpha_2(u) \rfloor \geq 1\}$. In terms of α we can write $W = \{u | \alpha(u) \geq 3\}$, as follows from the definition of α_2. Hence, W contains the users that are waiting for at least three complete sweeps.

If (5.13) holds for all $u \in V$, where $C = W$, then the request of each user from W is granted. Otherwise, the request of $u' \in W$ is granted if and only if

$$\min_{u \in V} y_u \cdot \frac{\lfloor \alpha_2(u') \rfloor}{\sum_{u \in W} \lfloor \alpha_2(u) \rfloor} \geq \frac{\max(\sigma_1(d(u')),0)}{sb_d} - \frac{\max(\sigma_1(i_{u'}),0)}{sb_d} \tag{5.14}$$

holds. The meaning is similar to the meaning of Equation (5.3). The right-hand side gives the contribution of user u' to the right-hand side of Equation (5.13) and for each user a part of $\min_{u \in V} y_u$ is reserved that is linear in the number of complete pair of sweeps the user is waiting for his/her request being granted.

Again, we do not want the choice of C to be affected by the choice of A. Let u' be a user whose request has been granted and who is waiting for a sign to start consuming. Adding u' to A does not affect the choice of C in the case that $u' \notin U$, because then $u' \notin V$. The choice is neither affected whenever (5.13) holds for $C = W$ or $\min_{u \in V} y_u \leq y_{u'}$

Next, we discuss the worst-case response time of the presented algorithm. By definition, a data block is large enough to survive two successive sweeps in which a data block is fetched for each user at a transfer time of t_d and the switch time is maximal. Consequently, if no data block is fetched for, say, user u, in two successive sweeps because the user is waiting for a request being granted, then $t_d + s_2(n) - s_2(n-1)$ time units are saved. Hence, during the sweeps the consuming users have consumed minimally

$$\frac{t_d + s_2(n) - s_2(n-1)}{sb_d} \tag{5.15}$$

data blocks less than would have been the case when the time that is reserved for user u in the two sweeps would have been spent for u. The request of u can arrive at the server just after the start of a sweep in which a data block is fetched for the user. Consequently, when the user is waiting for $2m$ sweeps, at least $2(m-1)$ times the number of data blocks given by (5.15) are saved. The increase of (2.20) and (2.21) resulting from granting the request of u, i.e. the right-hand side of (5.14) is at most $\max_{0 \leq i < n_B} \sigma_1(i)/sb_d$. Furthermore, it can be shown that $s_2(n) - s_2(n-1) \leq s(n) - s(n-1)$. Consequently, if a user waits $2 + 2\gamma_2$ sweeps, where

$$\gamma_2 = \left\lceil \frac{\max_{0 \leq i < n_B} \sigma_1(i)}{sb_d} \right\rceil, \tag{5.16}$$

then the number of data blocks that is saved during these sweeps is at least the increase of the minimum number of data blocks that have to be in the buffer at the start of the sweep resulting from granting the request of u, i.e. the increase of (2.20) and (2.21). The next lemma states that $2 + 2\gamma_2$ is indeed the maximum number of sweeps that is required for a request being granted.

Lemma 6. *Let $t_d < t(0)$ and $n \geq 2$. Then it takes at most $2 + 2\gamma_2$ sweeps before a request is granted, where γ_2 is defined by (5.16).*

Proof. We prove by induction to the number of executed sweeps, denoted by w, that just before C is determined, for all $u \in V$ either

$$y_u \geq \begin{cases} \sum_{u \in W} \lfloor \alpha_2(u) \rfloor \cdot \frac{t_d + s(n) - s(n-1)}{sb_d} & \text{if} \quad u \in X_{w-1} \\ \sum_{u \in W \cup V_2} \lceil \alpha_2(u) \rceil \cdot \frac{t_d + s(n) - s(n-1)}{sb_d} & \text{if} \quad u \notin X_{w-1}. \end{cases} \tag{5.17}$$

holds, where $V_2 = \{u | \alpha(u) = 2\}$, or (5.13) holds, where $C = W$. In the remainder of the proof we mean the case that $C = W$ when we refer to (5.13). We first prove that this proves the lemma.

Let u be a user who sends a request to the server. It takes at most one sweep before the next sweep starts. If it is a request to stop consuming, then the request is granted immediately. Because $t_d < t(0)$, $\gamma_2 \geq 1$. Hence, the lemma holds in the case that $d(u) = n_B$.

Assume that $d(u) < n_B$ and assume that the lemma does not hold. We show that this leads to a contradiction. Because the lemma does not hold, there is a sweep w, such that between the end of sweep w and the start of sweep $w+1$ $\alpha(u) = 1 + 2\gamma_2$ and $u \notin C$. Let u', $u' \in V$, be the user for whom $y_{u'}$ is minimal. We may assume that either (5.17) or (5.13) holds. Because $\gamma_2 \geq 1$, $u \in W$. Hence, if (5.13) holds, the request of u' is granted, which gives the contradiction. Assume that (5.17) holds. This implies that

$$\frac{y_u}{\sum_{v \in W} \lfloor \alpha_2(u) \rfloor} \geq \frac{t_d + s(n) - s(n-1)}{sb_d}.$$

Consequently, according to (5.14) the request of u' is granted if

$$\frac{t_d + s(n) - s(n-1)}{sb_d} \cdot \lfloor \alpha_2(u') \rfloor \geq \frac{\max(\sigma_1(d(u')),0)}{sb_d} - \frac{\max(\sigma_1(i_{u'}),0)}{sb_d}.$$

Because $\alpha(u') = 1 + 2\gamma_2$, $\lfloor \alpha_2(u') \rfloor = \gamma_2$. From the definition of γ_2, it follows that the previous equation is valid. Consequently, the request of u' is granted, which gives the contradiction.

We now prove by induction to the number of executed sweeps that just before C is determined either (5.17) or (5.13) holds. From the proof of Theorem 4, it follows that whenever y_u, $u \in V$, is non-negative at the start of a sweep, this is also the case at the end of the sweep. Furthermore, by the definitions of A and C, y_u does not become negative as a result of Algorithm 5. Consequently, $y_u \geq 0$ is invariant.

As basis of the induction we take the case $w = 0$. In that case, $V = \emptyset$ when C is determined. Hence, the induction hypothesis holds for $w = 0$.

Assume that $w = m+1$, $m \geq 0$. By the definition of A, (5.13) holds just before C is determined both if $V = A$ and if Equation (5.13) holds for all $u \in V \setminus A$. Assume that both cases do not hold. By the definition of A, there consequently exist a user u from $V \setminus A$, such that y_u is minimal. As long as the buffer of a user remains empty after the start-up of the system, $u \notin V$ holds, as follows from the definition of V and A. Consequently, because $u \notin A$, sweep $m+1$ can not be the first sweep, i.e. $m \geq 1$. By the induction hypothesis, we may assume that just before C is determined and m sweeps have been executed, either (5.17) or (5.13) holds. Similarly as in the proof of Lemma 5 it can be showed that if it holds at the end of sweep $m+1$, it also holds just before C is determined. Hence, proving that the induction hypothesis holds for $m+1$ comes down to proving that it holds just after the execution of sweep $m+1$ under the assumption that it holds just before C is determined between sweep m and $m+1$.

Assume that a data block has been fetched for u in sweep $m+1$. Similarly as in the proof of Lemma 5, it can be showed that just before the start of sweep $m+1$ Equation (5.17) holds. We prove that it also holds at the end of the sweep. We use the convention that if a tilde above a variable indicates that the value of is based on the situation just before the start of sweep $m+1$ and on the situation just after the end of sweep $m+1$, otherwise. As mentioned in the proof of Theorem 4, the number of data blocks in the buffer of u at the end of sweep $m+1$ is given by 1 plus the number of data blocks in the buffer at the start of the sweep minus (A.10). By the definition of y_u, we have that the buffer contains at least

$$1 + \frac{|X_m| \cdot t_d + s(|X_m|)}{sb_d} + \sum_{v \in \tilde{U}} \frac{\max(\sigma_1(\tilde{i}_v),0)}{sb_d} + \tilde{y}_u - \frac{|U_{m+1}| \cdot t_d + s(|U_{m+1}|)}{sb_d} - \sum_{v \in U_{m+1}} \frac{t_a(\tilde{i}_v) - t_d}{sb_d}$$

data blocks at the end of the sweep. Using that $U_{m+1} \subseteq X_m$ and the same arguments as in the proof of Theorem 4 after Expression (A.15), we get that the expression is at least

$$1 + \frac{|X_m \setminus U_{m+1}| \cdot t_d + s(|X_m|) - s(|U_{m+1}|)}{sb_d} + \sum_{v \in U} \frac{\max(\sigma_1(i_v),0)}{sb_d} + \tilde{y}_u.$$

By definition, $\alpha(u') \geq 3$, for all $u' \in W$. Furthermore, $\alpha(u') \geq 2$, $1 \leq u \leq n$, implies that $u' \in X_m$. Hence, $W \cup V_2 \subseteq X_m$ and $(W \cup V_2) \cap U_{m+1} = \emptyset$. Consequently, $|X_m \setminus U_{m+1}| \geq |W \cup V_2|$ and $s(|U_{m+1}|) \leq s(|X_m| - |W \cup V_2|)$, which implies that y_u is at least $\tilde{y}_u$ plus

$$\frac{|W \cup V_2| \cdot t_d + s(|X_m|) - s(|X_m| - |W \cup V_2|)}{sb_d}.$$

It can be showed that $s(|X_m|) - s(|X_m| - |W \cup V_2|) \geq |W \cup V_2| \cdot (s(n) - s(n-1))$. Because a data block is fetched for u in sweep $m+1$, no data block is fetched for u in sweep m. Hence, $u \notin X_{m+1}$ and $u \in X_m$. We have that (5.17) holds before the start of sweep $m+1$. To prove that it also holds at the end of the sweep it suffices to prove that

$$\frac{|W \cup V_2| \cdot (t_d + s(n) - s(n-1))}{sb_d} \geq \left(\sum_{v \in W \cup V_2} \lceil \alpha_2(v) \rceil - \sum_{v \in \tilde{W}} \lfloor \tilde{\alpha}_2(v) \rfloor \right) \cdot \frac{t_d + s(n) - s(n-1)}{sb_d}, \tag{5.18}$$

which can be written as $|W \cup V_2| \geq \sum_{u \in W \cup V_2} \lceil \alpha_2(u) \rceil - \sum_{u \in \tilde{W}} \lfloor \tilde{\alpha}_2(u) \rfloor$. It can be verified that $\tilde{W} \subseteq W$ and that $\alpha(u') = \tilde{\alpha}(u') + 1$, $u' \in W \cup V_2$. Hence, (5.18) holds if for all $u' \in W \cup V_2$

$$1 \geq \lceil \frac{\tilde{\alpha}(u')}{2} \rceil - |u' \in \tilde{W}| \cdot \lfloor \frac{\tilde{\alpha}(u') - 1}{2} \rfloor \tag{5.19}$$

holds. Let $u' \in W \cup V_2$. If $u' \notin \tilde{W}$, then, by the definition of W, $\tilde{\alpha}(u') = 2$. Hence, (5.19) is valid. Assume that $u' \in \tilde{W}$. If $\tilde{\alpha}(u') \bmod 2 = 0$, then (5.19) can be written as $1 \leq \frac{\tilde{\alpha}(u')}{2} - (\frac{\tilde{\alpha}(u')}{2} - 1)$, which is clearly true. If, on the other hand, $\tilde{\alpha}(u') \bmod 2 = 1$, then (5.19) equivales $1 \leq \frac{\tilde{\alpha}(u')}{2} + 1 - \frac{\tilde{\alpha}(u')}{2}$, which is also true. This completes the proof for the case that a data block is fetched for user u.

Assume that neither a data block is fetched for u in sweep m nor in sweep $m+1$. We get that $u \in X_m \cap X_{m+1}$ and $u \notin U_{m+1}$. By the definition of filling strategy DS and because, by assumption, $u \in V$, u has room for less than one data block in the buffer at the start of sweep $w+1$, i.e. u has at least $1 + n \cdot \max_{1 \leq i < n_B} \sigma_1(i)/sb_d$ data blocks in the buffer. A lowerbound for the number of data blocks in the buffer at the end of the sweep is given by this number minus (A.10). Hence, the buffer of u contains at least

$$1 + n \cdot \max_{1 \leq i < n_B} \frac{\sigma_1(i)}{sb_d} - \frac{|U_{m+1}| \cdot t_d + s(|U_{m+1}|)}{sb_d} - \sum_{v \in U_{m+1}} \frac{t_a(\tilde{i}_v) - t_d}{sb_d}$$

data blocks. By the same arguments as from Expression (A.12), we can rewrite to expression to

$$\frac{(n - |U_{m+1}|) \cdot t_d + s_2(n) - s(|U_{m+1}|)}{sb_d} + (n - |U_{m+1}|) \cdot \max_{1 \leq i < n_B} \frac{\sigma_1(i)}{sb_d} + \sum_{v \in U_{m+1}} \frac{\max(\sigma_1(i_v), 0)}{sb_d}.$$

From the definition of X, it follows that $X_{m+1} \cap U_{w+1} = \emptyset$. Consequently, $n - |U_{m+1}| \geq |X_{m+1}|$ and $s_2(n) - s(|U_{m+1}|) \geq s(|X_{m+1}|)$. By the definition of y_u we consequently have that

$$y_u \geq |X_m| \cdot \max_{0 \leq i < n_B} \frac{\sigma_1(i)}{sb_d} - \sum_{v \in X_{m+1} \cap U} \frac{\max(\sigma_1(i_v), 0)}{sb_d}.$$

Because $W \subseteq X_{m+1}$, this implies (5.13).

Finally, we consider the case that a data block is fetched for u in sweep m and no data block is fetched in sweep $m+1$. Hence, $u \notin X_m$ and $u \in X_{m+1}$. By the induction hypothesis, we have that just before C is determined and m sweeps have been executed, either (5.17) or (5.13) holds. Again, we can prove in a similar way as in the proof of Lemma 5, that just before the start of sweep $m+1$ equation (5.17) holds. By the definition of y_u this means that the buffer of u contains at the start of sweep $m+1$ at least

$$1 + \sum_{v \in \tilde{U}} \frac{\max(\sigma_1(\tilde{i}_v), 0)}{sb_d} + \sum_{v \in \tilde{W} \cup \tilde{V}_2} \lceil \tilde{\alpha}_2(v) \rceil \cdot \frac{t_d + s(n) - s(n-1)}{sb_d}$$

data blocks. At the end of the sweep the buffer contains at least this number minus (A.10). Similar as from Expression (A.12), we can prove that this equals

$$\frac{(n-|U_{m+1}|)\cdot t_d + s_2(n) - s(|U_{m+1}|)}{sb_d} + \sum_{v\in U} \frac{\max(\sigma_1(i_v),0)}{sb_d} + \sum_{v\in \tilde{W}\cup\tilde{V}_2} \lceil \tilde{\alpha}_2(v)\rceil \cdot \frac{t_d + s(n) - s(n-1)}{sb_d}.$$

By the definition of y_u and because $u \in X_{m+1}$, proving

$$\sum_{v\in \tilde{W}\cup\tilde{V}_2} \lceil \tilde{\alpha}_2(v)\rceil \cdot \frac{t_d + s(n) - s(n-1)}{sb_d} \geq \sum_{v\in W} \lfloor \alpha_2(v)\rfloor \cdot \frac{t_d + s(n) - s(n-1)}{sb_d}$$

suffices to prove that (5.17) holds at the end of sweep $m+1$. Because $\tilde{W} \subseteq W$ and because $\alpha(u') = \tilde{\alpha}(u) + 1$, $u \in W$, the equation is true if for all $u' \in W$, it holds that

$$|u' \in \tilde{W}\cup\tilde{V}_2| \cdot \lceil \frac{\tilde{\alpha}(u')-1}{2}\rceil \geq \lfloor \frac{\tilde{\alpha}(u')}{2}\rfloor. \tag{5.20}$$

If $u' \notin \tilde{W}\cup\tilde{V}_2$, then it follows from the definition of W that $\tilde{\alpha}(u') \leq 1$. Hence, (5.20) holds. Assume that $u' \in \tilde{W}\cup\tilde{V}_2$. If $\tilde{\alpha}(u') \bmod 2 = 0$, then both the left hand side and the right hand side of (5.20) equal $\frac{\tilde{\alpha}(u')}{2}$. If, on the other hand, $\tilde{\alpha}(u') \bmod 2 = 1$, then they both equal $\frac{\tilde{\alpha}(u')-1}{2}$. Consequently, (5.20) holds. □

Whenever an arbitrary user u has at least $1 + n\cdot \max_{0\leq i<n_B} \sigma_1(i)/sb_d$ data blocks in the buffer, then (5.13) holds for $C = W$. which, by the definition of A, implies that u may start consuming. Because, by the used filling strategy, only one data block can be fetched in two successive sweeps, it takes $1 + 2\eta_2$ sweeps to fetch this number of data blocks, where

$$\eta_2 = \lceil n \cdot \max_{0\leq i<n_B} \frac{\sigma_1(i)}{sb_d}\rceil. \tag{5.21}$$

The following theorem combines this observation with Lemma 6.

Theorem 18. *Let $t_d < t(0)$ and $n \geq 2$. Then the worst-case response time is bounded by time that is maximally required for $3 + 2\gamma_2 + 2\eta_2$ successive sweeps, where γ_2 and η_2 are defined according to (5.16) and (5.21).*

Proof. By the definition of filling strategy DS, the sweep in which the first requested data block of u if fetched must be preceded by a sweep in which no data block is fetched for u. At most one sweep after the arrival of the request of u at the server, the next sweep starts. From this moment, no data block is fetched for u, until the request is granted. Consequently, after two sweeps the first requested data block of u may be fetched as far as the used filling strategy is concerned. From Lemma 6, it follows that it takes at most $2 + 2\gamma_2$ sweeps before the request of, say, user u is granted. Because $2 + 2\gamma_2 \geq 2$, it takes at most $2 + 2\gamma_2$ sweeps before the start of, say, sweep w in which the first requested data block can be fetched.

To prove the theorem, it now suffices to show that it takes at most $1 + 2\eta_2$ sweeps from sweep w before u can start consuming. We prove it by contradiction. Hence, assume that user u is still not consuming at the start of sweep $w + 2\eta_2 + 1$, while user u wants to watch a movie. Because the buffers are large enough to store $1 + \eta_2$ data blocks, $1 + \eta_2$ data blocks are fetched for u during sweep w through $w + 2\eta_2$. Furthermore, $u \in X_{w+2\eta_2}$ at the end of sweep $w + 2\eta_2$. From the definition of η_2 and y_u it now follows that (5.13) holds for $C = W$. Hence, $u \in A$ between sweep $w + 2\eta_2$ and $w + 2\eta_2 + 1$, which gives the contradiction. □

In order to express the worst-case response time in time units, we investigate the time that is worst-case required for, say, m successive sweeps. By the used filling strategy, the worst-case time for two successive sweeps is $n\cdot t(0) + s_2(n)$. Furthermore, the worst-case time for one sweep is $n\cdot t(0) + s(n)$. The next theorem states that if m is divisible by two, then the time that is maximally required for m successive sweeps is given by $\frac{m}{2}$ times the time that is maximally required for two successive sweeps, i.e. the time required for each successive pair of sweeps is maximal. For the case that m is not divisible by two, the

theorem only states an upperbound on the time that is maximally required for m successive sweeps. The upperbound is given by the time that is maximally required for $m-1$ sweeps, which is $\frac{m-1}{2}$ times the time that is maximally required for two successive sweeps because $m-1$ is even, plus the time that is maximally required for one sweep.

Theorem 19. *The time that is maximally required for m successive sweeps equals* $\frac{m}{2} \cdot (n \cdot t(0) + s_2(n))$ *if m is divisible by two and is bounded by* $\frac{m-1}{2} \cdot (n \cdot t(0) + s_2(n)) + n \cdot t(0) + s(n)$ *if m is not divisible by two.*

Proof. Let w be an arbitrary sweep. We investigate the time that is maximally required for the sweeps w through $w+m-1$. Assume that m is divisible by two. By the definition of filling strategy DS, at most one data block can be fetched per user in two successive sweeps. A possible scenario is that in the sweeps $w, w+2, \ldots, w+m-2$ a data block is fetched for the users 1 through $\lceil \frac{n}{2} \rceil - 1$ and in the sweeps $w+1, w+3, \ldots, w+m-1$ a data block is fetched for the other users. By the same arguments as we used in the proof of Theorem 14, we can show only the data block is fetched that is stored on position 0. This and (1.1) gives that the required time for the m sweeps can be $\frac{m}{2} \cdot (n \cdot t(0) + s_2(n))$. Clearly, the time is also an upperbound on the time required for m successive sweeps.

Assume that m is not divisible by two. Because $m-1$ is divisible by two, the time required for the first $m-1$ sweeps is maximal $\frac{m-1}{2} \cdot (n \cdot t(0) + s_2(n))$, as we showed above. Furthermore, the time required for the mth sweep does not exceed $n \cdot t(0) + s(n)$. Hence, the time required for m sweeps does not exceed $\frac{m-1}{2} \cdot (n \cdot t(0) + s_2(n)) + n \cdot t(0) + s(n)$. This time is not a strict upperbound, because the only scenario according to which $\frac{m-1}{2} + 1$ data blocks are fetched for each user in m sweeps, is the scenario that a data block is fetched for each user in the sweeps $w, w+2, \ldots, w+m-1$ and no data block is fetched in the sweeps $w+1, w+3, \ldots, w+m-2$. As a result, the required time equals $(\frac{m-1}{2} + 1) \cdot (n \cdot t(0) + s(n))$, which is strictly less than the derived upperbound because $s_2(n) > s(n)$. □

In the next example we illustrate the meaning of the discussed theorems.

Example 2 revisited By Theorem 18 the worst-case response time is bounded by $3 + 2\gamma_2 + 2\eta_2$ sweeps. By definition, $\gamma_2 = \lceil \frac{x}{t_d + s(n) - s(n-1)} \rceil$. Because $s(n) = 109.45$ ms and $s(n-1) = 101.17$ ms, $\gamma_2 = \lceil 0.2223 \rceil = 1$. Furthermore, $\eta_2 = \lceil 0.2095 \rceil = 1$. Hence, the worst-case response time is seven sweeps. By Theorem 19, the time required for seven successive sweeps is bounded by $\frac{m-1}{2} \cdot (n \cdot t(0) + s_2(n)) + n \cdot t(0) + s(n)$, which equals 1.65 seconds.

For DS the worst-case response time is the time that is maximally required for three successive sweeps. From the proof of Theorem 19, it follows that the theorem also holds for DS. Hence, the time required for three successive sweeps does not exceed $n \cdot t_{\min} + s_2(n) + n \cdot t_{\min} + s(n)$, where $t_{\min} = \frac{308.49}{r_{\min}}$. Hence, the worst case response time does not exceed 1.19 seconds. As a result, the worst-case response time increases by 39%. □

Chapter 6

Results

In this thesis we presented an approach for utilizing the different transfer rates of the disk, such that the relation between the block size and the maximum number of admitted users can be based on a higher transfer rate than the minimum transfer rate, as is the case in TB and DS. It was thereby our aim to minimize the cost per user. In this chapter we quantify for a practical example the reduction in the cost per user resulting from our approach in relation to TB and DS. Furthermore, we compare the cost per user of our approach with track pairing [1]. Before we discuss the results in Section 6.2, we discuss track pairing in more detail in Section 6.1.

6.1 Track Pairing

We first discuss track pairing, as it is presented in [1]. At the end of the section we give an improvement of the approach. As discussed in Section 1.2, the length of a track grows linearly with its distance to the spindle. Furthermore, a disk rotates at a constant angular velocity. Consequently, by conceptually pairing the innermost track with the outermost one, the second innermost with the second outermost, and so on, both the total length of each pair of tracks as their read time is constant (see Figure 6.1). Although the length of a track grows linearly with its distance to the spindle, the capacity and, correspondingly, the transfer rate do not, because in the same zone each track has the same capacity. However, the capacity per track in a zone does grow linearly with the distance of the zone to the spindle. Consequently, the capacity and the transfer rate of a pair of tracks comes within a few percent of a constant. We denote the minimum transfer rate of any pair of tracks by r^{tp}_{min}. Hence, $r^{tp}_{min} = r_{avg} - \varepsilon$, where ε is a small positive number.

Hence, when a movie is recorded alternately on a range of contiguous outer tracks and their inner counterparts, the guaranteed transfer rate when reading a data block is r^{tp}_{min}. Consequently, the transfer time that is maximally required for reading a data block is given by $\frac{B}{r^{tp}_{min}}$. However, because the data blocks are split into two, a sweep in which n data block have to be fetched contains $2n$ disk accesses. Consequently, the time required for a sweep in which a data block is fetched for each user is maximally

$$n \cdot \frac{B}{r^{tp}_{min}} + s(2n)$$

time units. As proved for TB [11], safeness is guaranteed if filling strategy TB is used, a data block is large enough to survive one worst-case sweep and the buffers have room for three data blocks. Hence, if filling strategy TB is used, if the block size satisfies

$$B \geq c_{max}(n \cdot \frac{B}{r^{tp}_{min}} + s(2n)) \tag{6.1}$$

and if the buffers have room for three data blocks, then safeness is guaranteed. This algorithm will be

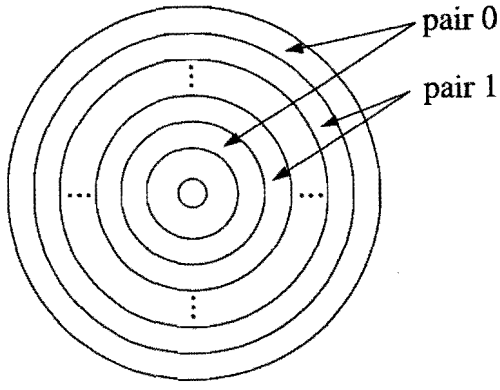

Figure 6.1. Track Pairing

denoted by TB^{tp}. Because $r_{min}^{tp} \approx r_{avg}$, the relation between the block size and the maximum number of admitted users is based on approximately the average transfer rate of the disk instead of the minimum transfer rate, as was the case for the original TB algorithm. However, this is at the cost of a larger worst-case switching overhead. Hence, whether or not the required buffer size for a given number of users decreases in relation to TB, depends on whether the gain in the required worst-case transfer time exceeds the increase in the worst-case switching overhead.

As proved for DS, when filling strategy DS is used, safeness is guaranteed if a data block is large enough to survive two sweeps and the buffers have room for two data blocks. In the case that track pairing is applied, the condition on the block size can be formalized as

$$B \geq c_{max}(n \cdot \frac{B}{r_{min}^{tp}} + s_2(2n)). \tag{6.2}$$

We denote this algorithm by DS^{tp}.

As a result of applying track pairing, the maximum number of users for whom a video-on-demand system can be designed with a given hard disk increases, as we will show. The Equations (6.1) and (6.2) can be fulfilled if and only if $c_{max} \cdot n < r_{min}^{tp}$. Consequently, by using track pairing we can design our video-on-demand system for each number of users up to $\frac{r_{min}^{tp}}{c_{max}}$. From Section 1.4.1 and 1.4.2, it follows that for the original TB and DS algorithm, this number is bounded by $\frac{r_{min}}{c_{max}}$, which is generally less.

When the discussed approach is slightly altered, the block size can be based on the average transfer rate instead of r_{min}^{tp}. Instead of pairing complete tracks, we pair the tracks such that the total length of a pair is still two tracks but the part from the inside and the outside have not necessarily an equal size. Moreover, the tracks are paired such that each pair has an equal capacity, which equals two times the average capacity of a track. Consequently, each pair has an equal transfer rate. It can be proved that such a subdivision always exists. However, we will not give the proof in this thesis. Storing the data blocks on contiguous pairs yields a guaranteed transfer rate of r_{avg} instead of r_{min}^{tp}. Using this altered version of track pairing, the conditions on the block size for TB^{tp} and DS^{tp} become

$$B \geq c_{max}(n \cdot \frac{B}{r_{avg}} + s(2n)) \tag{6.3}$$

and

$$B \geq c_{max}(n \cdot \frac{B}{r_{avg}} + s_2(2n)), \tag{6.4}$$

respectively.

6.2 Comparison of the approaches

In this section we compare the cost per user resulting from our approach with track pairing and the original TB and DS algorithms, where the minimum transfer rate is assumed. Because the cost per user only differ in the relation between the maximum number of admitted users and the minimum required buffer size, we discuss this relation.

We define DA1 as the disk scheduling algorithm, where filling strategy TB is used, the buffer sizes are given by Theorem 1, Algorithm 1 is used for minimizing the buffer sizes and Algorithm 3 is used for handling the requests. Similarly, we define DA2 and DADS as the disk scheduling algorithms which are based on Theorems 3 and 4, respectively.

As stated in [11], TB is safe if and only if there is room for three data blocks. However, this is not exactly the case as can be inferred as follows. We showed in Section 2.1.1 that safeness is guaranteed if a buffer is large enough to guarantee that whenever no data block is fetched for the user in a given sweep w, the user has at least one data block in the buffer at the start of sweep $w+1$. If no data block is fetched for a user in a sweep w, then at most $(n-1)$ data blocks are fetched in that sweep. Consequently, the time required for the sweep is bounded by $(n-1)\cdot t_{max} + s(n-1)$. Dividing this by the time that can minimally be survived with a data block, i.e., $n \cdot t_{max} + s(n)$, gives the number of data blocks that is maximally consumed during the sweep. Hence, if no data block is fetched for a user, which means that the buffer has room for strictly less than one data block, the buffer has to contain at least one data block plus this number. As a result, safeness is guaranteed if each buffer has room for at least

$$2 + \frac{(n-1)\cdot t_{max} + s(n-1)}{n \cdot t_{max} + s(n)}$$

data blocks, which can be rewritten to

$$3 - \frac{t_{max} + s(n) - s(n-1)}{n \cdot t_{max} + s(n)}. \tag{6.5}$$

We will assume this buffer size for TB. Similarly, the minimum buffer size for TB^{tp} is given by

$$3 - \frac{t_{avg} + s(2n) - s(2(n-1))}{n \cdot t_{avg} + s(2n)}. \tag{6.6}$$

data blocks, instead of three. We now discuss how for each disk scheduling algorithm the minimum buffer size can be determined for a given maximum number of admitted users.

For TB and TB^{tp} the minimum buffer size for a given number of users is given by the minimum block size multiplied by (6.5) and (6.6), respectively. The block size for these algorithms is minimal whenever equality holds in (1.2) and (6.3), respectively. Analogously, for DS and DS^{tp} the minimum buffer size is given by two times the block size for which (1.3) and (6.4) are equalities. As for TB, TB^{tp}, DS and DS^{tp}, the buffer requirements in DA2 are minimized once the block size is minimized. Clearly, the minimum block size will be at most the minimum block size given by TB. The minimum block size can be calculated by checking for an increasing block size whether (2.10) holds for a_2. We can not use binary search, because a different block size possibly results in an essentially different composition of the set of positions. Hence, although for a particular block size a_2 does not satisfy (2.10), this can be the case when the block size is decreased.

For DA1 and DADS the required buffer size is not necessarily minimized once the block size is minized because the number of data blocks a buffer has to contain generally increases when the block size decreases. Again, we calculate the minimum buffer size by comparing the buffer requirements for an increasing block size.

By Theorem 9, we can start at a block size that is based on the average transfer rate of the disk, i.e., for DA1 and DA2 we can start at the block size that satisfies $B = c_{max}(n \cdot \frac{B}{r_{avg}} + s(n))$ and for DADS we can

Zone	Transfer rate (Mbit/s)	Capacity (MByte)
0	59	598
1	64	776
2	70	1163
3	77	1363
4	83	1606
5	89	1080
6	92	2285

Table 6.1. Arrangement of the zones

start at the block size that satisfies $B = c_{max}(n \cdot \frac{B}{r_{avg}} + s_2(n))$. For DA2, we stop whenever a block size is encounterend for which a_2 satisfies (2.10) or when we consider a block size as not interesting anymore, because it has become too large. The second criterion is also used for DA2 and DADS. A buffer must be at least large enough to store one data block, because otherwise no data block can be fetched for a user. Consequently, when the block size exceeds the buffer size that has already been derived to be safe, the program can stop as well. Some stricter upperbounds can be derived, but we will not discuss this problem in this thesis.

We have analyzed the cost per user for the case that a single Seagate Barracuda 9 drive is used (see Table 1.1) and the users consume at a bit rate that varies between 0 and 4 Mbit/s. The arrangement of the zones is given in Table 6.1. Table 6.2 and Figure 6.2 show the results for the different disk scheduling algorithms that are based on filling strategy TB. In Table 6.3 and Figure 6.3 this is done for disk scheduling algorithms that are based on filling strategy DS. In Tables 6.2 and 6.3 the columns denoted by 'Mem' give the buffer size per user in MByte. The columns denoted by '%Mem' give the buffer size per user as percentage of the buffer size per user for TB in Table 6.2 and as percentage of the buffer size per user for DS in Table 6.3.

It follows that the cost per user for (especially) DA1 and DADS are considerably smaller than for TB and TB^{tp} and DS and DS^{tp}, respectively. For example, if maximally 10 users have to be serviced, the buffer requirements in DA1 are roughly one third of the buffer requirements given by TB and almost half of the buffer requirements given by TB^{tp}.

It can be read from the tables that at most 16 users can be serviced with the disk. As mentioned, with TB^{tp}, DA1, DS^{tp} and DADS any number up to $\frac{r_{avg}}{c_{max}}$ can be serviced with a given disk. However, it follows from (6.1) that $r_{avg} = 78.6$ Mbit/s and $c_{max} = 4$ MBit/s. Consequently, $\frac{r_{avg}}{c_{max}} = 19.65$, which is considerably larger than 16. The reason is that we combined in this thesis the transfer rate, the track switch time and the head switch time into one (lower) transfer rate, while Table 6.1 gives the real transfer rate.

In Tables 6.4 and 6.5, the worst-case response times are presented for the different disk scheduling algorithms. In the tables, the column 'resp' denotes the worst-case response time and the column '%resp' gives the worst-case response time as percentage of the worst-case response time of TB and DS. The results are visualized in Figure 6.4 and 6.5. From the tables and figures it follows that although the worst-case response times for DA1, DA2 and DADS are higher than for the other disk scheduling algorithms, they are still reasonable for many applications.

	TB		TB^{tp}		DA1		DA2	
n	Mem	%Mem	Mem	%Mem	Mem	%Mem	Mem	%Mem
5	0.219	100	0.195	89	0.116	53	0.124	57
6	0.290	100	0.261	90	0.145	50	0.153	53
7	0.395	100	0.327	83	0.192	49	0.197	50
8	0.518	100	0.405	78	0.233	45	0.243	47
9	0.709	100	0.517	73	0.287	40	0.313	44
10	1.029	100	0.632	61	0.364	35	0.399	39
11	1.578	100	0.797	51	0.455	29	0.498	32
12	2.973	100	1.020	34	0.579	19	0.674	23
13	11.668	100	1.313	11	0.764	7	0.956	8
14			1.779		1.042		1.470	
15			2.538		1.505		2.641	
16			4.045		2.521		10.445	
17			8.910		5.845			

Table 6.2. The buffer size per user (in MBytes) as a function of the maximum number of users for the case that filling strategy TB is used.

	DS		DS^{tp}		DADS	
n	Mem	%Mem	Mem	%Mem	Mem	%Mem
5	0.154	100	0.154	100	0.107	69
6	0.203	100	0.199	98	0.125	62
7	0.275	100	0.244	89	0.155	56
8	0.359	100	0.299	83	0.191	53
9	0.490	100	0.375	77	0.235	48
10	0.709	100	0.455	64	0.290	41
11	1.084	100	0.574	53	0.353	33
12	2.037	100	0.729	36	0.462	23
13	7.980	100	0.934	12	0.588	7
14			1.256		0.796	
15			1.777		1.167	
16			2.883		1.921	
17			6.242		4.553	

Table 6.3. The buffer size per user (in MBytes) as a function of the maximum number of users for the case that filling strategy DS is used.

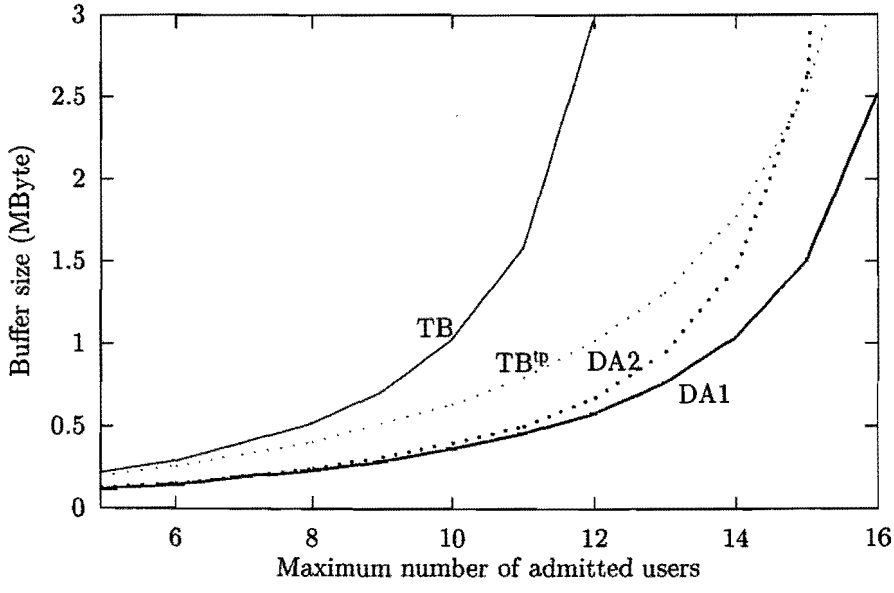

Figure 6.2. Relation between maximum number of admitted users and the buffer requirements in the case that filling strategy TB is used

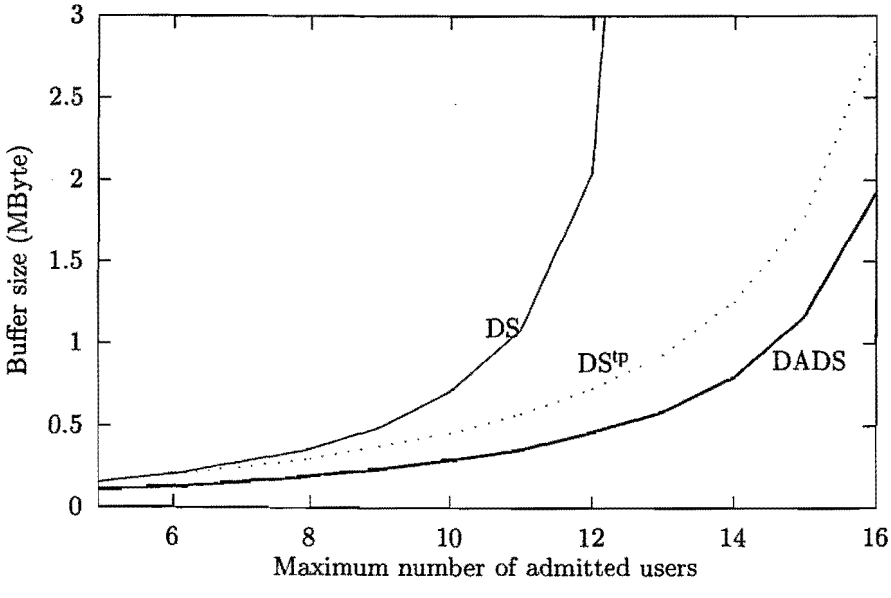

Figure 6.3. Relation between maximum number of admitted users and the buffer requirements in the case that filling strategy DS is used

	TB		TB^{tp}		DA1		DA2	
n	resp	%resp	resp	%resp	resp	%resp	resp	%resp
5	0.22	100	0.19	86	0.34	155	0.35	159
6	0.30	100	0.26	87	0.43	143	0.43	143
7	0.43	100	0.33	77	0.57	133	0.57	128
8	0.58	100	0.41	71	0.71	122	0.71	117
9	0.81	100	0.55	68	0.87	107	0.87	106
10	1.21	100	0.68	56	1.10	91	1.10	91
11	1.89	100	0.89	47	1.39	74	1.36	72
12	3.63	100	1.18	33	1.78	49	1.84	51
13	14.49	100	1.56	11	2.35	16	2.61	18
14			2.17		3.25		4.00	
15			3.18		4.73		7.17	
16			5.21		7.96		28.32	
17			11.80		16.96			

Table 6.4. The worst-case response time per user (in seconds) as a function of the maximum number of users for the case that the buffer requirements are given by Table 6.2

	DS		DS^{tp}		DADS	
n	resp	%resp	resp	%resp	resp	%resp
5	0.25	100	0.29	116	0.42	168
6	0.32	100	0.38	119	0.51	159
7	0.45	100	0.46	102	0.65	144
8	0.60	100	0.57	95	0.80	133
9	0.85	100	0.72	85	0.97	114
10	1.27	100	0.88	69	1.24	98
11	2.00	100	1.11	56	1.53	77
12	3.89	100	1.42	37	2.01	52
13	15.76	100	1.83	12	2.56	16
14			2.47		3.46	
15			3.51		5.12	
16			5.72		8.26	
17			12.44		18.31	

Table 6.5. The worst-case response time per user (in seconds) as a function of the maximum number of users for the case that the buffer requirements are given by Table 6.3

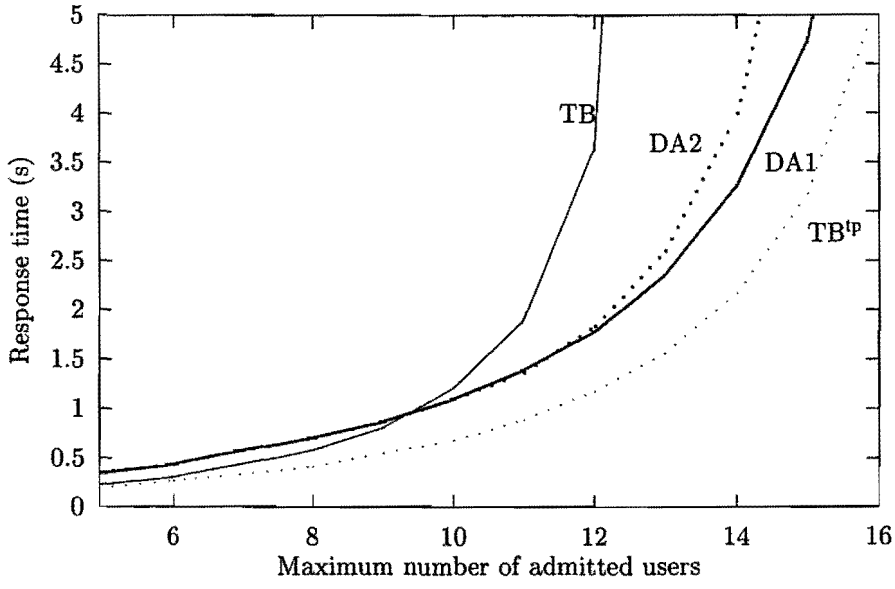

Figure 6.4. Relation between maximum number of admitted users and the worst-case response time in the case that the buffer requirements are given by Table 6.2

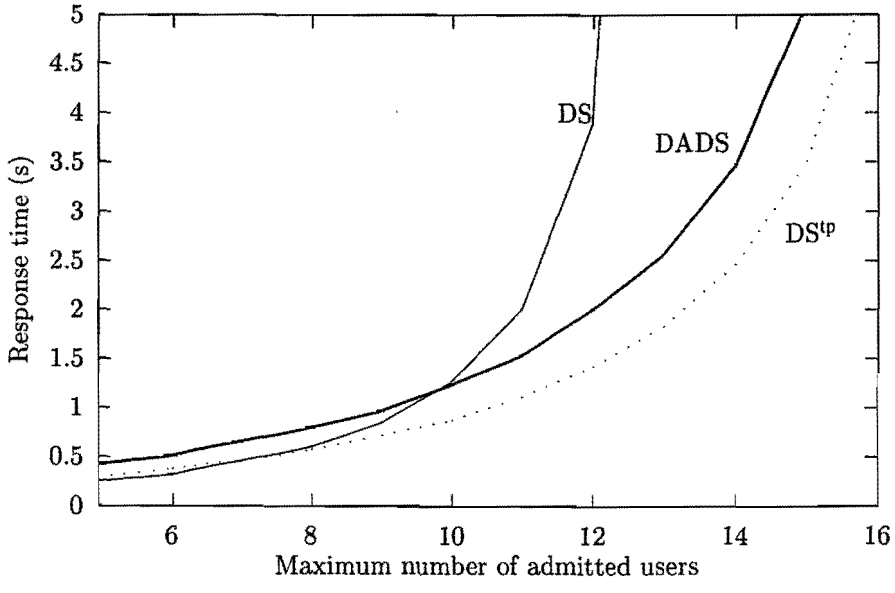

Figure 6.5. Relation between maximum number of admitted users and the worst-case response time in the case that the buffer requirements are given by Table 6.3

Chapter 7

Generalization to multiple disks and more than one movie

In this thesis we assumed that the collection of movies is stored on a single hard disk. However, when more, possibly different, disks are used, both the total storage capacity and the throughput are increased. Several approaches are presented in literature for increasing the guaranteed throughput of the disk array, for example striping [2] and random duplicated assignment (RDA) [9]. In Section 7.1 we show how our approach for utilizing the different transfer rates of a single disks can be combined with full striping. Throughout this thesis, we assumed that only one single movie is stored on the hard disk and that the movie covers the entire disk. In Section 7.2, we show that these assumptions are not restrictive, i.e., we show that whenever the video-on-demand system is designed under these assumptions, it is also possible to store a set of movies that does not necessarily cover the entire disk.

7.1 Multi-disk model

In full striping each data block is striped across all disks in the disk array. Hence, if the video server contains m disks, then each data block is partitioned into m sub-blocks, where the ith sub-block is stored on disk i. Furthermore, all sub-blocks have the same size.

We first show how TB and DS can be adapted for the case that striping is used. As proved for TB [11], safeness is guaranteed if a data block is large enough to survive a worst-case sweep, if filling strategy TB is used and if a buffer is large enough to store three data blocks. When striping is used, at most one sub-block per user has to be fetched in a sweep. Hence, the time that is maximally required for a sweep is given by $n \cdot \frac{b}{r_{\min}} + s(n)$, where b is the size of a sub-block, i.e., $b = \frac{B}{m}$. Consequently, safeness is guaranteed if the block size satisfies

$$B \geq c_{\max}(n \cdot \frac{\frac{B}{m}}{r_{\min}} + s(n)),$$

if filling strategy TB is used and if a buffer is large enough to store three data blocks.

Likewise, it follows from the safeness of DS that safeness of filling strategy DS is guaranteed in combination with striping if the block size satisfies

$$B \geq c_{\max}(n \cdot \frac{\frac{B}{m}}{r_{\min}} + s_2(n))$$

and a buffer is large enough to store two data blocks.

Before we discuss how the disk scheduling algorithms DA1, DA2 and DADS can be used in combination with full striping, we show that we may assume that each disk can store an equal number of data blocks and we impose requirements on the way that the data is striped across the disks. Let i be the disk with minimum capacity and assume that there is a disk j, such that $n_B^i < n_B^j$, where the index indicates the disk for which the variable is defined. Because in full striping data is equally distributed over all disks, at any moment at least $n_B^j - n_B^i$ positions of disk j are left unused. For the sake of a high throughput, it is preferred to let the inner positions of the disk unused. As a result, we can consider disk j as a disk with only n_B^i positions, namely the outer n_B^i positions. Hence, we can assume that each disk can store an equal number of data blocks, denoted by n_B. We define the numbering of the positions on each disk similarly as in Section 1.3, i.e., we define the numbering as depicted in Figure 1.4.

In the original form, the position to which a sub-block is assigned is independent of the positions to which the other sub-blocks of the same data block are assigned. However, we now assign each data block on positions with the same number. Hence, $a^i(x^i) = a^j(x^j)$ for all $1 \leq i \leq j \leq m$ and $x \in L$. Consequently, we can omit the index in the assignment.

We define a single disk on which the algorithms are based, such that the time required for a sweep in the system with the virtual single disk is an upperbound on the time required for a sweep in the system with the disk array, where the described striping technique is applied. If the difference between these times is compensated by idle time[1], then the behaviour of the system with the single disk is equivalent to the behaviour of the system with the disk array. As a result, safeness is guaranteed when we use DA1, DA2 or DADS and base the disk scheduling algorithm on the defined imaginary single disk.

We define the transfer time function t, such that $t(x)$ gives the maximum required transfer time required for transferring a sub-block from position x. Hence, $t(x) = \max_{1 \leq i \leq m} t^i(x^i)$. By definition, the transfer rate on position x, i.e., $r(x)$, is now given by $\frac{B}{t(x)}$. Because $t(x)$ is the transfer time required for reading $\frac{1}{m}$th data block, $r(x) = m \cdot \min_{1 \leq i \leq m} r^i(x^i)$. Similarly, we define $s(j)$ as the maximum switch time required for fetching j data blocks in a sweep. Formally, $s(j) = \max_{1 \leq i \leq m} s^i(j)$. Consider a sweep in which the data blocks $y_1, y_2, \ldots, y_l$, $0 \leq l \leq n$, have to be fetched in the multi-disk model. The time required for the sweep is given by the maximum required time of any disk. Hence, the sweep takes maximally

$$\max_{1 \leq i \leq m} \left(\sum_{j=1}^{l} t_a^i(y_j^i) + s^i(l) \right) \tag{7.1}$$

time units. We have to show that this time is bounded by the time that is required for the sweep in our single disk model in which the same data blocks have to be fetched. In the single disk model the time required for the sweep is given by

$$\sum_{j=1}^{l} t_a(y_j) + s(l).$$

Using the definitions of t and s this can be written to

$$\sum_{j=1}^{l} \max_{1 \leq i \leq m} t_a^i(y_j^i) + \max_{1 \leq i \leq m} s^i(l).$$

This is clearly at most (7.1). As a result, if DA1, DA2 or DADS is used and if the algorithm is based on the virtual single disk, then safeness is guaranteed. As mentioned at the end of Section 4.2, if a single disk is used, then our approach can be used to design the system for all users up to $\frac{r_{avg}}{c_{max}}$ and for each block size that is based on maximally the average transfer rate. However, if different disks are used, then the value of r_{avg} depends on the composition of the set of positions on each disk and correspondingly on the block size.

[1] The introduction of idle time is not necessary to guarantee safeness. However, if we do not introduce idle time, then the proves of some theorems have to be changed slightly because the behaviours of the imaginary disk-model and the multi-disk model are not equivalent anymore.

Hence, we can not give such a statement anymore. Nevertheless, because generally the value of r_{avg} will be within a few percent of a constant, the statement will still approximately be valid.

7.2 Storing more than one movie

So far, we assumed that only one single movie is stored on the hard disk and that the movie consists of exactly n_B data blocks. We will show that these assumptions are not restrictive. Hence, we show that whenever the video-on-demand system is designed under these assumptions, we can also store multiple movies on the disk that not necessarily cover the entire disk, such that safeness remains guaranteed. We first introduce some notation.

Let m be the number of movies that have to be stored on the disk. We number these movies from 0 through $m-1$. Furthermore, we define n_B^i as the number of data block of movie i and number these data blocks from 0^i through $n_B^i - 1$. As a result, the set of data blocks, denoted by L^i, is defined as $\{0^i, 1^i, \ldots, n_B^i - 1\}$. Finally, we define $a^i : L^i \rightarrow F$ as the injective function that indicates how the movie is stored on the hard disk. Note that a^i is no longer a bijection.

In order to show that multiple movies can be stored on a disk, such that safeness is guaranteed, we give conditions under which the multiple movies can be stored such that safeness is guaranteed and we show that it is always possible to satisfy these conditions. Assume that the buffer sizes are based on assignment a. Before we discuss the conditions under which multiple movies can be stored, we discuss under which conditions safeness is guaranteed when the single movie that covers the entire disk is stored according to another assignment, say, a'.

Let the buffer sizes be given by Theorem 1. Assume that the assignment a' satisfies

$$\max_{0 \leq i < n_B} \sigma_1(a',i) \leq \max_{0 \leq i < n_B} \sigma_1(a,i) \text{ and } \max_{0 \leq i < n_B - 1} \sigma_2(a',i) \leq \max_{0 \leq i < n_B - 1} \sigma_2(a,i), \tag{7.2}$$

where $\sigma_m(a,i)$ is defined as $\sigma_m(i)$, where the assignment to which $\sigma_m(i)$ is related is parameterized. Because Theorem 1 states that a buffer size larger than (2.7) is safe as well, (7.2) implies that the assumed buffer sizes and, correspondingly, Algorithm 3 are also safe if the movie is stored according to assignment a' instead of a.

Assume that the buffer size is given by Theorem 3. From the theorem, it follows that this buffer size is safe for any assignment that satisfies (2.10). Hence, if a' satisfies (2.10), then safeness is guaranteed for the assumed buffer size in combination with Algorithm 4.

Similarly as in the case of Theorem 1, it can be verified that if the buffer sizes are given by Theorem 4, then safeness is guaranteed in the case that the movie is stored according to assignment a' when

$$\max_{0 \leq i < n_B} \sigma_1(a',i) \leq \max_{0 \leq i < n_B} \sigma_1(a,i).$$

This inspires us to the following conditions on the assignments of the multiple movies, such that safeness is guaranteed when the buffer requirements are given by Theorems 1, 3 and 4 and the requests are handled by the corresponding algorithms. If buffer sizes are given by Theorem 1, then safeness is guaranteed when for each assignment a^j, $0 \leq j < m$,

$$\max_{0 \leq i < n_B^j} \sigma_1(a^j,i) \leq \max_{0 \leq i < n_B} \sigma_1(a,i) \text{ and } \max_{0 \leq i < n_B^j - 1} \sigma_2(a^j,i) \leq \max_{0 \leq i < n_B - 1} \sigma_2(a,i) \tag{7.3}$$

holds. If the buffer requirements are given by Theorem 3, then safeness is guaranteed when each assignment a^j, $0 \leq j < m$, satisfies

$$t_d \geq \frac{\max_{0 \leq i < n_B^j - 1}(t_{a^j}(i) + t_{a^j}(i+1))}{2} \tag{7.4}$$

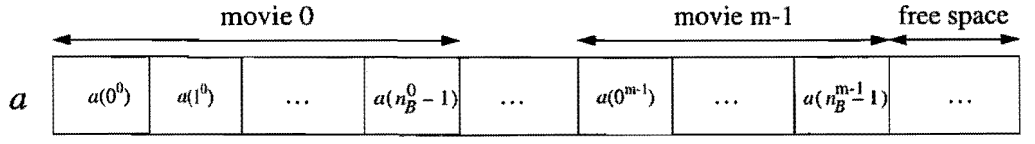

Figure 7.1. Assignments for multiple movies

and in the case that the buffer requirements are given by Theorem 4 safeness is guaranteed when

$$\max_{0 \le i < n_B^j} \sigma(a^j, i) \le \max_{0 \le i < n_B} \sigma(a, i) \tag{7.5}$$

for all a^j, $0 \le j < m$.

We formalize the second case in the next theorem. For the proof of the theorem, we refer to Appendix A. The other two statements can be proved by the same strategy. However, we will not include the proves in this thesis.

Theorem 20. *If for all i, $0 \le i < m$, (7.4) holds, filling strategy TB is used and the requests of the users are handled by Algorithm 4, then safeness is guaranteed if the buffers have room for $3 - \delta_2$ data blocks, with δ_2 defined by (2.12).* □

We still have to show that it is possible to satisfy the given conditions for the multiple movies. This is the case when the assignments are defined as depicted in Figure 7.1.

Finally, we briefly discuss the maintenance of the video-on-demand system. In the case that the buffer requirements are given by Theorems 1, 3 or 4, it can be the case that almost the only way to store a given collection of movies on disk such that safeness is guaranteed, is as depicted in Figure 7.1, i.e. by letting each assignment a^i be a subsequence of a. When this is the case, frequently adding and deleting files can lead to a considerable fragmentation of the disk space. This yields a considerable loss of storage capacity, which can only be avoided by defragmentation. Therefore, it may be preferred that the buffer requirements are not fully minimized, such that it is easier to fulfill Equations (7.3), (7.4) and (7.5).

Chapter 8

Conclusion

In a video-on-demand system, the video data is often stored on hard disks, due to their large storage capacity and the possibility of random access. The system offers the users a continuous data stream by periodically placing a data block in the buffer of each user. A disk scheduling algorithm is used for scheduling the disk accesses, such that the buffers neither underflow nor overflow. In addition, the disk scheduling algorithm also has to handle user requests. The choice of the disk scheduling algorithm depends on whether the users consume at a variable or constant bit rate and whether data blocks of constant or variable size are being retrieved. In this thesis we focussed on the case that users consume at a variable bit rate and the data blocks are of constant size. Furthermore, we mainly focussed on the case that the movies are stored on a single disk. Although we assumed that only one movie is stored on the disk that covers the entire disk, we showed that these assumptions are not restrictive.

The cost per user of the system is affected by the disk scheduling algorithm by defining a relation between the maximum number of admitted users and the block size and by giving a minimum number of data blocks that have to fit in the buffer. In most disk scheduling algorithms presented in literature, the relation between the maximum number of admitted users and the block size is determined by the guaranteed throughput of the disk. Nowadays, a hard disk often consists of several zones that each have a different transfer rate. As a result, the guaranteed throughput differs considerably from the average throughput of the disk. In this thesis we focussed on defining a placement of the data blocks of the movie on the hard disk, such that a higher throughput can be guaranteed over a period of time. In addition, we revised the buffer requirements of two disk scheduling algorithms, namely the triple buffering algorithm and the dual sweep algorithm [11], such that the relation between the maximum number of users and the size of a data block can be based on this higher guaranteed throughput. With the placement and the revision we aimed at minimizing the cost per user.

In the triple buffering algorithm and the dual sweep algorithm, a request of a user is granted immediately after the request arrives at the server. However, when we allow this, then it can be the case that the users only request data blocks from the inner zone, which implies that we cannot guarantee a higher throughput than the minimum throughput. Therefore, we assumed that no requests are sent by the user for designing the video-on-demand system and we investigated how requests can be handled such that the system remains safe.

First, we revised the buffer requirements of the triple buffering algorithm and the dual sweep algorithm, such that the block size can be based on a higher throughput than the minimum throughput. We derived a sufficient buffer size for the case that the same strategy for scheduling the disk accesses is used as in the triple buffering algorithm and that the relation between the maximum number of admitted users and the block size is based on the guaranteed throughput over a period in which maximally two data blocks are fetched per user. Furthermore, we derived a sufficient buffer size for the case that the second condition

does not hold and we derived a necessary and sufficient buffer size for the case that the same strategy is used as in dual sweep.

Except for the first case, called the conditional solution, the number of data blocks that have to fit in the buffer increases with the maximum cumulative difference between the actual transfer time and the dimension time, i.e., the transfer time on which the block size is based, over any subset of successive data blocks of the movie. We proved that the problem of finding an assignment such that the maximum is minimized is NP-hard in the strong sense. We relaxed the problem to finding an assignment for which the number of subsets over which we have to maximize is minimized. We proved that if the dimension time is at least the average transfer time of any, say, k_{ub} successive data blocks, then we only have to maximize over all subsequences with length at most k_{ub}. Consequently, we relaxed the problem to finding an assignment for which k_{ub} is minimized. We showed that this problem is still NP-hard. Furthermore, we proved that the relaxation does not lead to an assignment if the dimension time is smaller than the average transfer time over all data blocks of the movie.

The approach we presented to solve the relaxed problem is that, for an increasing k_{ub}, we try to find an assignment for which the dimension time is at least the average transfer time of any k_{ub} successive data blocks, until it is found. For a given k_{ub} this comes down to minimizing the maximum sum of the transfer times of k_{ub} successive data blocks and checking whether the maximum exceed k_{ub} times the dimension time. Hence, our new problem is to minimize the total sum of the transfer times of a given number of successive data blocks.

Because we assumed that the size of a data block is based on a higher throughput than the minimum throughput, we only have to consider the case that k_{ub} is at least two. For $k_{ub} = 2$, we gave an optimal assignment, i.e., an assignment for which the maximum sum of the transfer times of two successive data blocks is minimal. This also optimally solves the problem of finding an assignment, such that the buffer requirements given by the conditional solution are safe. For larger values of k_{ub} we presented an heuristic.

We showed that for each dimension time larger than the average transfer time, our approach to solve the relaxed problem outputs a feasible assignment, i.e., it finds a k_{ub} and an assignment, such that the average sum of any k_{ub} successive data blocks does not exceed the dimension time. However, as a result of using a heuristic for minimizing the sum of the transfer times of three or more successive data blocks, k_{ub} does not have to be minimal.

For each of the three derived and minimized buffer requirements, we defined an algorithm for handling user requests. For each of these algorithms, we gave the worst-case response time. The combination of the buffer requirements and the algorithms for handling requests gives three disk scheduling algorithms, namely DA1, DA2 and DADS.

We evaluated the results by means of a practical example. Thereby we compared DA1, DA2 and DADS with the original triple buffering algorithm and the original dual sweep algorithm and with these algorithms combined with track pairing [1]. The required buffer size for a given number of users is reduced considerably by our disk scheduling algorithms in comparison with the other algorithms. Furthermore, more users can be serviced with a given buffer size. Consequently, our disk scheduling algorithms result in a considerable reduction in the cost per user. The reduction is at the cost of a higher worst-case response time. However, the worst-case response time is still reasonable for many applications.

Finally, we also discussed how our disk scheduling algorithms can be used in combination with full striping. Despite of this, more research can be done in combining our theory with several other techniques for storing and retrieving data from a disk array. Secondly, some more attention can be paid on handling requests, which we did not discuss thoroughly because our focus was on reducing the cost per user. Especially in the field of the average response times some improvements can be achieved. We briefly discussed the problem of adding and deleting movies. The problem how this can be done best is an interesting research area, as well. Moreover, some research can be done in basing our theory on other disk scheduling algorithms than the triple buffering algorithm and the dual sweep algorithm. Our theory is for example also applicable to n-EDFC [11].

Appendix A

Proofs buffer capacities

Theorem 1. *If $t_d \geq t(n_B - 1)$, where $t_d = \frac{B - c_{max} \cdot s(n)}{c_{max} \cdot n}$, if no requests are sent by the users and if each user from U has initially at least the number of data blocks given by (2.6) in the corresponding buffer, then filling strategy TB is safe if there is room for at least*

$$3 + n \cdot \max_{0 \leq i < n_B - 1} \frac{\sigma_2(i)}{sb_u} - \delta \tag{A.1}$$

data blocks, where σ_m and δ are defined by (2.3) and (2.8), respectively.

Proof. Because a data block is only fetched for a user if the buffer has room for it, no buffer overflow occurs. Furthermore, buffer underflow can only occur for users from U, as follows from its definition. Assume that at the start of a given sweep w, the buffer of each user from U contains at least the number of data blocks given by (2.6). By assumption, this holds initially. For proving that no buffer underflow occurs, it suffices to show for an arbitrary $u \in U$ (1) that the number of data blocks given by (2.6) fits in the buffer of u, (2) that it is sufficient to survive sweep w and (3) that at the end of the sweep the buffer contains (again) at least the number of data blocks given by (2.6), where U and i_u, $1 \leq u \leq n$, have been updated. Updating U comes down to removing user u, $1 \leq u \leq n$, from U if and only if data block $n_B - 1$, i.e., the last data block of the movie, has been fetched for u in sweep w. Updating i_u comes down to an increase by one if and only if a data block has been fetched for user u in sweep w. The proof that the number of data blocks given by (2.6) fits in a buffer is discussed later on. We first prove that it is sufficient to survive sweep w.

Proof of (2):
Expression (2.6) is at least $1 + \sum_{u \in U} \frac{t_a(i_u) - t_d}{sb_u} + (n - |U|) \cdot \frac{\tau}{sb_u}$, as follows from the definition of σ_1. By definition, sb_u is the time that can minimally be survived with a single data block. Hence, the buffer of u contains enough data to survive

$$sb_u + \sum_{u \in U} (t_a(i_u) - t_d) + (n - |U|) \cdot \tau \tag{A.2}$$

time units. By definition, $sb_u = n \cdot t_d + s(n)$ and $\tau = t(n_B - 1) - t_d$. Consequently, (A.2) can be rewritten to $\sum_{u \in U} t_a(i_u) + (n - |U|) \cdot t(n_B - 1)$. From $n \geq |U|$ and $t(n_B - 1) \geq 0$, it follows that the expression is at least $\sum_{u \in U} t_a(i_u) + s(n)$. Because the users from U are the only users, for whom possibly a data block is fetched in sweep w, this expression gives the time that is maximally required for sweep w. Consequently, no buffer underflow occurs during sweep w.

Proof of (3):
Proving that at the end of sweep w the buffer of u contains at least the number of data blocks given by (2.6), where U and $i_{u'}$, $1 \leq u' \leq n$, have been updated, comes down to proving that if $u \in U \setminus \{u | u \in D \wedge i_u =$

$n_B - 1\}$, i.e., if the last data block of the movie is not fetched for u in sweep w, then the buffer contains at least

$$1 + \sum_{\substack{u' \in D \\ i_{u'} \neq n_B - 1}} \frac{\sigma_1(i_{u'}+1)}{sb_u} + \sum_{u' \in U \setminus D} \frac{\sigma_1(i_{u'})}{sb_u} + (n - (|U| - |\{u | u \in D \wedge i_u = n_B - 1\}|)) \cdot \frac{\tau}{sb_u} \tag{A.3}$$

data blocks at the end of sweep w, where D is defined as the set of users, for whom a data block is fetched in sweep w. Hence,

$$D = \{u | u \in U \wedge \text{the buffer of } u \text{ has room for at least one data block}\}.$$

Consequently, the time required for w is bounded by $\sum_{u' \in D} t_a(i_{u'}) + s(|D|)$. Dividing by sb_u gives the maximum number of data blocks that are consumed during sweep w. For proving that the buffer of u contains at least the number of data blocks given by (A.3) at the end of the sweep, we distinguish the cases that $u \in D$ and $u \notin D$.

Assume that $u \in D$. The minimum filling of the buffer of u at the end of the sweep, expressed in data blocks, is given by the initial content, which is given by (2.6) minus the maximum number of data blocks that can be consumed during the sweep plus the fetched data block. Hence, the buffer contains at least

$$1 + \sum_{u' \in U} \frac{\sigma_1(i_{u'})}{sb_u} + (n - |U|) \cdot \frac{\tau}{sb_u} - \frac{\sum_{u' \in D} t_a(i_{u'}) + s(|D|)}{sb_u} + 1$$

data blocks. We split the range of the u' in the second term. Furthermore, we use that $\sum_{u' \in D} t_a(i_{u'})$ equals $\sum_{u' \in D}(t_a(i_{u'}) - t_d) + |D| \cdot t_d$ and 1 equals $\frac{n \cdot t_d + s(n)}{sb_u}$. This yields

$$1 + \sum_{\substack{u' \in D \\ i_{u'} \neq n_B - 1}} \frac{\sigma_1(i_{u'})}{sb_u} + \sum_{\substack{u' \in D \\ i_{u'} = n_B - 1}} \frac{\sigma_1(i_{u'})}{sb_u} + \sum_{u' \in U \setminus D} \frac{\sigma_1(i_{u'})}{sb_u} - \frac{\sum_{u' \in D}(t_a(i_{u'}) - t_d)}{sb_u}$$
$$+ \frac{n \cdot t_d + s(n) - |D| \cdot t_d - s(|D|)}{sb_u} + (n - |U|) \cdot \frac{\tau}{sb_u}.$$

We put the fifth term together with the second and third terms and use that $\sigma_1(i_{u'}) = t_a(i_{u'}) - t_d$, if $i_{u'} = n_B - 1$. This gives that the buffer of u contains at least

$$1 + \sum_{\substack{u' \in D \\ i_{u'} \neq n_B - 1}} \left(\max_{1 \leq k \leq n_B - i_{u'}} \frac{\sum_{j=0}^{k-1}(t_a(i_{u'}+j) - t_d)}{sb_u} - \frac{(t_a(i_{u'}) - t_d)}{sb_u} \right) + \sum_{u' \in U \setminus D} \frac{\sigma_1(i_{u'})}{sb_u}$$
$$+ \frac{(n - |D|) \cdot t_d + s(n) - s(|D|)}{sb_u} + (n - |U|) \cdot \frac{\tau}{sb_u}$$

data blocks. By definition, $u' \in U$ implies $i_{u'} < n_B$. Hence, the domain of k in the second term is not empty. Consequently, we can simplify the second term to $\sum_{u' \in D} \max_{1 \leq k \leq n_B - i_{u'}} \frac{\sum_{j=1}^{k-1}(t_a(i_{u'}+j) - t_d)}{sb_u}$. Applying the dummy transformations $j := j + 1$ and $k := k - 1$ on this term yields

$$1 + \sum_{\substack{u' \in D \\ i_{u'} \neq n_B - 1}} \max_{0 \leq k \leq n_B - (i_{u'}+1)} \frac{\sum_{j=0}^{k-1}(t_a(i_{u'}+1+j) - t_d)}{sb_u} + \sum_{u' \in U \setminus D} \frac{\sigma_1(i_{u'})}{sb_u} + \frac{(n - |D|) \cdot t_d + s(n) - s(|D|)}{sb_u}$$
$$+ (n - |U|) \cdot \frac{\tau}{sb_u}. \tag{A.4}$$

A restriction of the range of k will not increase the value of the maximum. As a result, we can omit $k = 0$ from the range of k in the second term without increasing the value of the complete expression. Because the cardinality of D is at most n and s is ascending, as follows from its meaning, $s(|D|) \leq s(n)$. Furthermore, $t_d \geq 0$. Consequently, deleting the fourth term does not increase the value of the expression. From the

assumption $t_d \geq t(n_B - 1)$, it follows that $\tau \leq 0$. As a result, $(n - |U|) \cdot \tau \geq (n - |U| + |\{u | u \in D \wedge i_u = n_B - 1\}|) \cdot \tau$. This completes the proof of (A.3) for the case that $u \in D$.

Assume that $u \notin D$. By the used filling strategy, this implies that the buffer of u has no room for a data block at the start of sweep w. Hence, by (A.1), the buffer contains at least $2 + n \cdot \max_{0 \leq i < n_B - 1} \sigma_2(i)/sb_u$ data blocks at the start of sweep w. Because the time required for sweep w is at most $\sum_{u' \in D} t_a(i_{u'}) + s(|D|)$ and because the time that can minimally be survived with a single data block is sb_u, the buffer of u contains at the end of sweep w at least

$$2 + n \cdot \max_{0 \leq i < n_B - 1} \frac{\sigma_2(i)}{sb_u} - \delta - \frac{\sum_{u' \in D} t_a(i_{u'}) + s(|D|)}{sb_u}$$

data blocks, which is at least

$$2 + \sum_{\substack{u' \in D \\ i_{u'} \neq n_B - 1}} \frac{\sigma_2(i'_u)}{sb_u} + \sum_{\substack{u' \in D \\ i_{u'} = n_B - 1}} \frac{\sigma_2(i'_u - 1)}{sb_u} + (n - |D|) \cdot \max_{0 \leq i < n_B - 1} \frac{\sigma_2(i)}{sb_u} - \delta - \frac{\sum_{u' \in D} (t_a(i_{u'}) - t_d)}{sb_u}$$
$$- \frac{|D| \cdot t_d + s(|D|)}{sb_u}.$$

Similarly as above, we put the sixth term together with the second and third terms. This gives

$$2 + \sum_{\substack{u' \in D \\ i_{u'} \neq n_B - 1}} \frac{\sigma_1(i_{u'} + 1)}{sb_u} + \sum_{\substack{u' \in D \\ i_{u'} = n_B - 1}} \frac{t_a(i_{u'} - 1) - t_d}{sb_u} + (n - |D|) \cdot \max_{0 \leq i < n_B - 1} \frac{\sigma_2(i)}{sb_u} - \delta - \frac{|D| \cdot t_d + s(|D|)}{sb_u}.$$

Writing down the definition of δ and writing 1 as $\frac{n \cdot t_d + s(n)}{sb_u}$ yields

$$1 + \sum_{\substack{u' \in D \\ i_{u'} \neq n_B - 1}} \frac{\sigma_1(i_{u'} + 1)}{sb_u} + \sum_{\substack{u' \in D \\ i_{u'} = n_B - 1}} \frac{t_a(i_{u'} - 1) - t_d}{sb_u} + (n - |D| - 1) \cdot \max_{0 \leq i < n_B - 1} \frac{\sigma_2(i)}{sb_u} + \max_{0 \leq i < n_B} \frac{\sigma_1(i)}{sb_u}$$
$$+ \frac{(n - |D| - 1) \cdot t_d - s(|D|) - s(n) + s(n-1) + s(n)}{sb_u}. \tag{A.5}$$

From $u \notin D$ it follows that the cardinality of D is strictly less than n. This and the ascendingness of s yield that $s(|D|)$ is at most $s(n-1)$. Hence, $-s(|D|) - s(n) + s(n-1) + s(n) \geq 0$. As a result, we can delete this expression from the numerator of the last term without increasing the value of the complete expression.

Next, we prove that

$$\max_{0 \leq i < n_B - 1} \sigma_2(i) + t_d \geq \max_{0 \leq i < n_B} \sigma_1(i). \tag{A.6}$$

Let i_0 be the block number for which $\sigma_1(i)$ is maximal. We distinguish the cases $i_0 < n_B - 1$ and $i_0 = n_B - 1$. Assume that $i_0 < n_B - 1$. From the definition of σ_2 and σ_1 it follows that $\sigma_1(i_0) = \max(\sigma_2(i_0), t_a(i_0) - t_d)$. Consequently, because $t_d \geq 0$ Equation (A.6) is implied by

$$\sigma_2(i_0) + t_d \geq t_a(i_0) - t_d. \tag{A.7}$$

From $i_0 < n_B - 1$ we obtain that $\sigma_2(i_0) \geq \sum_{j=0}^{1} (t_a(i_0 - 1 + j) - t_d)$. Hence, Equation (A.7) follows from

$$\sum_{j=0}^{1} (t_a(i_0 + j) - t_d) + t_d \geq t_a(i_0) - t_d.$$

This can be written as $t_a(i_0 + 1) \geq 0$, which holds because the range of t_a is positive.

Assume that $i_0 = n_B - 1$. Then, $\sigma_1(i_0) = t_a(i_0) - t_d$ and $\sigma_2(i_0 - 1) \geq \sum_{j=0}^{1} (t_a(i_0 - 1 + j) - t_d)$. Hence, (A.6)

is implied by

$$\sum_{j=0}^{l}(t_a(i_0-1+j)-t_d)+t_d \geq t_a(i_0)-t_d.$$

This can be rewritten to $t_a(i_0-1) \geq 0$, which is true. As a result, Equation (A.6) holds.

Using (A.6) we get that (A.5) is at least

$$1+\sum_{\substack{u' \in D \\ i_{u'} \neq n_B-1}} \frac{\sigma_1(i_{u'}+1)}{sb_u}+\sum_{\substack{u' \in D \\ i_{u'}=n_B-1}} \frac{t_a(i_{u'}-1)-t_d}{sb_u}+(n-|D|)\cdot \max_{0\leq i<n_B} \frac{\sigma_1(i)}{sb_u}. \tag{A.8}$$

Because $\max_{0\leq i<n_B} \sigma_1(i) \geq \sigma_1(i_{u'})$, $1 \leq u' \leq n$, this expression is at least

$$1+\sum_{\substack{u' \in D \\ i_{u'} \neq n_B-1}} \sigma_1(i_{u'}+1)+\sum_{u' \in U \setminus D} \sigma_1(i_{u'})+\sum_{\substack{u' \in D \\ i_{u'}=n_B-1}} \frac{t_a(i_{u'}-1)-t_d}{n\cdot t_d+s(n)}+(n-|U|)\cdot \max_{0\leq i<n_B} \sigma_1(i).$$

From the definition of τ and the descendingness of t it follows that the numerator in the fourth term as well as $\max_{0\leq i<n_B} \sigma_1(i)$ are at least τ. Hence, the expression is at least (A.3).

Proof of (1):
As mentioned in the beginning of the proof, we still have to prove that the number of data blocks given by (2.6) fits in a buffer. We have just proved that, if there is room for less than one data block at the start of a sweep, the buffer contains at least the number of data blocks given by (A.3) at the end of the sweep, regardless of the composition of D. We thereby did not use the assumption that the buffer contains initially the number of data blocks given by (2.6). Taking $D = \emptyset$ gives that at the end of the sweep the buffer contains at least $1+\sum_{u=1}^{n} \sigma(i_u)/sb_u$ data blocks. Hence, it fits in the buffer. □

Theorem 4. *If $n \geq 2$ and $t(0) \geq t_d \geq t(n_B-1)$, where $t_d = \frac{B-c_{max}\cdot s_2(n)}{c_{max}\cdot n}$, if the users do not send requests to the server and if each user from U has initially at least the number of data blocks given by (2.21) in the buffer, then filling strategy DS is safe if and only if there is room for at least*

$$2+n\cdot \max_{0\leq i<n_B} \frac{\sigma_1(i)}{sb_d} \tag{A.9}$$

data blocks, where σ_m is defined by (2.3).
Proof. We first prove the sufficient condition. By the used filling strategy, a data block is only placed in a buffer whenever there is room for it. Hence, no buffer overflow occurs. We prove that neither buffer underflow occurs. Let u' be an arbitrary user from U. Assume that at the start of a given sweep w the buffer of u' contains at least the number of data block given by (2.21) if $u' \notin U_w$ and at least the number given by (2.20) otherwise. By assumption, this holds initially. To prove that no buffer underflow occurs it suffices to prove (1) that the number of data blocks given by (2.20) and (2.21) fit in the buffer of u', (2) that it is sufficient to survive sweep w and (3) that at the end of the sweep a buffer contains (again) at least the number of data blocks given by (2.21) if $u' \notin U_w$ and at least the number given by (2.20) otherwise, where w, i_u and U are updated in (2.20) and (2.21), i.e. w is replaced bye $w+1$, i_u is raised by one if and only if $u \in U_w$ and u is removed from U if and only if the last data block of the movie has been fetched for u in sweep w. Clearly, condition (1) is satisfied.

Proof of (2):
The time required for sweep w is maximally $\sum_{u\in U_w} t_a(i_u)+s(|U_w|)$, which can be written as $|U_w|\cdot t_d+s(|U_w|)+\sum_{u\in U_w}(t_a(i_u)-t_d)$. Dividing it by sb_d gives the number of data blocks that is maximally consumed during this time. Hence, during sweep w maximally

$$\frac{|U_w|\cdot t_d+s(|U_w|)}{sb_d}+\sum_{u\in U_w} \frac{t_a(i_u)-t_d}{sb_d} \tag{A.10}$$

data blocks are consumed. This is clearly less than (2.20) and (2.21). Hence, the number of data blocks given by (2.20) and (2.21) is sufficient to survive sweep w.

Proof of (3):
We have to prove that at the end of sweep w the buffer of u' contains at least the number of data blocks given by (2.21) if $u' \notin U_{w+1}$ and at least the number given by (2.20) otherwise, where w, i_u and U are updated in (2.20) and (2.21). We distinguish the cases that $u' \notin U_w$ and that $u' \in U_w$.

Assume that $u' \notin U_w$. We will prove that if $u' \notin U_{w+1}$, then (2.21) holds and that if $u' \in U_{w+1}$, then (2.20) holds. We start with the first case. So assume that $u' \notin U_w$ and $u' \notin U_{w+1}$. This implies that the buffer of u' does not have room for a data block at the start of sweep $w+1$, because otherwise a data block is fetched for u' in that sweep $w+1$, as follows from the definition of filling strategy DS. Consequently, the buffer of u' contains at least

$$1 + n \cdot \max_{0 \le i < n_B} \frac{\sigma_1(i)}{sb_d} \tag{A.11}$$

data blocks at the start of sweep $w+1$. By assumption, $t_d \le t(0)$. Hence, $\max_{0 \le i < n_B} \sigma_1(i) \ge 0$. Consequently, (A.11) is at least (2.21).

Assume that $u' \notin U_w$ and that $u' \in U_{w+1}$. We have to prove that (2.20) holds at the end of sweep w. By assumption, the buffer of u' contains at the start of sweep w at least the number of data blocks given by (2.21). Furthermore, we have derived that the number of data blocks that is maximally consumed during sweep w is given by (A.10). Hence, at the end of sweep w the buffer of u' contains at least

$$1 + \sum_{u \in U} \frac{\max(\sigma_1(i_u), 0)}{sb_d} - \frac{|U_w| \cdot t_d + s(|U_w|)}{sb_d} - \sum_{u \in U_w} \frac{t_a(i_u) - t_d}{sb_d} \tag{A.12}$$

data blocks. Because $U_w \subseteq U$, this can be rewritten to

$$\frac{(n - |U_w|) \cdot t_d + s_2(n) - s(|U_w|)}{sb_d} + \sum_{u \in U \setminus U_w} \frac{\max(\sigma_1(i_u), 0)}{sb_d}$$
$$+ \sum_{u \in U_w} \left(\frac{\max(\sigma_1(i_u), 0)}{sb_d} - \frac{t_a(i_u) - t_d}{sb_d} \right).$$

Because at most one data block is fetched for each user in two successive sweeps $n - |U_w| \ge |U_{w+1}|$ and $s_2(n) - s(|U_w|) \ge s(|U_{w+1}|)$. Hence, the first term in the previous expression is at least the first term in Expression (2.20). Because i_u does not change if $u \notin U_w$ and because i_u is raised by one if $u \in U_w$, proving that the previous expression is at least (2.20) comes down to showing that

$$\sum_{u \in U_w} (\max(\sigma_1(i_u), 0) - t_a(i_u) - t_d) \ge \sum_{u \in U_w \cap \tilde{U}} \max(\sigma_1(i_u + 1), 0), \tag{A.13}$$

where $\tilde{U}$ is U at the end of sweep w, i.e. $\tilde{U} = U \setminus \{u | u \in U_w \wedge i_u = n_B - 1\}$. By the definition of σ_m, the left-hand side of (A.13) equals

$$\sum_{u \in U_w} \left(\max \left(\max_{1 \le k \le n_B - i_u} \sum_{j=0}^{k-1} (t_a(i_u + j) - t_d), 0 \right) - t_a(i_u) - t_d \right). \tag{A.14}$$

Because $u \in U_w$ implies that $i_u < n_B$, $\max_{1 \le k \le n_B - i_u} \sum_{j=0}^{k-1} (t_a(i_u + j) - t_d)$ is at least $t_a(i_u) - t_d$. Hence, replacing the 0 in the maximum operator by $t_a(i_u) - t_d$ does not increase its value. When we also put $t_a(i_u) - t_d$ inside the maximum operator, we get that (A.14) is at least

$$\sum_{u \in U_w} \max \left(\max_{1 \le k \le n_B - i_u} \sum_{j=0}^{k-1} (t_a(i_u + j) - t_d) - (t_a(i_u) - t_d), 0 \right).$$

Putting $t_a(i_u) - t_d$ inside the sum-quantification, applying the dummy transformation $j = j+1$ and using the definition of σ_m give that the expression equals $\sum_{u \in U_w} \max(\sigma_1(i_u+1),0)$. If $u \in \tilde{U} \setminus U$, then $i_u = n_B - 1$. Hence, $\sigma_1(i_u+1) = -\infty$. Consequently, the contribution of u to the sum quantification is 0. Hence, the sum quantification equals the right-hand side of (A.13), which had to be proved.

Assume that $u' \in U_w$. Because at most one data block is fetched for each user in two successive sweeps, we have to prove that at the end of sweep w the buffer of u' contains the number of data blocks given by (2.21). We may thereby assume that the buffer contains at the start of the sweep at least the number of data blocks given by (2.20). We have already showed that the number of data blocks that is maximally consumed during sweep w is given by (A.10). Hence, at the end of the sweep the number of data blocks that the buffer of u' contains is at least 1 plus (2.20) minus (A.10). This means that the buffer contains at least

$$1 + \sum_{u \in U} \frac{\max(\sigma_1(i_u),0)}{sb_d} - \sum_{u \in U_w} \frac{t_a(i_u) - t_d}{sb_d} \tag{A.15}$$

data blocks. This expression equals

$$1 + \sum_{u \in U \setminus U_w} \frac{\max(\sigma_1(i_u),0)}{sb_d} + \sum_{u \in U_w} \left(\frac{\max(\sigma_1(i_u),0)}{sb_d} - \frac{t_a(i_u) - t_d}{sb_d} \right).$$

As proved above, the last quantification is at least $\sum_{u \in U_w} \max(\sigma_1(i_u+1),0)/sb_d$. This completes the proof that in the case that $u' \in U_w$ and $u' \notin U_{w+1}$, u' has at least the number of data blocks given by (2.21) in the buffer at the end of sweep w. This completes the proof of the sufficient condition.

Proof of necessary condition:
We prove the necessary condition by contradiction. Assume that the buffer of each user has room for $2 + n \cdot \max_{0 \le i < n_B} \frac{\sigma_1(i)}{sb_d} - x$ data blocks for some $x > 0$. We will prove that buffer underflow can occur. Let i_0, k_0 be such that $\sum_{j=0}^{k_0-1} (t_a(i_0+j) - t_d)$, i.e. $\sigma_1(i_0)$, is maximal. Suppose that at the start of a given sweep w all users are about to fetch data block i_0. Furthermore, there are $\lceil \frac{n}{2} \rceil$ users that have room for at least $1 + \varepsilon$ data blocks, $0 < \varepsilon \le \frac{n}{2} \cdot \frac{t(n_B-1)}{sb_d}$ and $\varepsilon < x$, and there are $\lfloor \frac{n}{2} \rfloor$ users that have room for less than one and more than $1 - \varepsilon$ data blocks. We define G_{even} and G_{odd} as the sets that contain the users of the first and second category, respectively. Finally, we assume that for none of the users from G_{even} a data block has been fetched in sweep $w-1$ and that all users consume at maximum bit rate. That the described situation can occur can be inferred as follows. If all users have room for less than one data block, no sweep will be executed. If the buffers of the users are inspected again, it can be the case that the buffers of some users have a little more room than one data block. Hence, the above described situation occurs.

We now prove the property that alternately data blocks are fetched for users from G_{even} and G_{odd}. Formally, if $p \bmod 2 = 0$, where p is an arbitrary integer between 0 and $2 \cdot k_0 - 1$, then a data block is fetched for all users from G_{even} in sweep $w+p$ and if $p \bmod 2 = 1$ then a data block is fetched for all users from G_{odd}. We prove this property, denoted by $Q(p)$ by induction. As basis we take $p = 0$ and $p = 1$. Because, by assumption, all users of G_{even} have at the start of sweep w room for at least 1 data block and there has not been fetched a data block for these users in sweep $w-1$, a data block is fetched for all these users in sweep w. Consequently, $Q(0)$ holds. By assumption, we also have that the users from G_{odd} do not have room for a data block at the start of sweep w. Hence, there will not be fetched a data block for them. The time required for sweep w is $|G_{\text{even}}| \cdot t_a(i_0) + s(|G_{\text{even}}|)$. Dividing by sb_d gives the number of data blocks that is consumed during this sweep, because, by assumption, the users consume at maximum bit rate. When we use that the cardinality of G_{even} is $\lceil \frac{n_B}{2} \rceil$ we get that during sweep w $\frac{\lceil \frac{n}{2} \rceil \cdot t_a(i_0) + s(\lceil \frac{n}{2} \rceil)}{sb_d}$ data blocks are consumed. This is strictly larger than the upperbound of ε. Hence, there are more than ε data blocks consumed. As a result, the users of G_{odd} have room for more than one data block at the start of sweep $w+1$. Consequently, there will be fetched a data block for all these users in sweep $w+1$. Hence, $Q(1)$ holds as well.

Next, we prove the induction step. Let p be any integer between 2 and $2 \cdot k_0 - 1$. We have to prove that $Q(p)$ holds under the assumption that $Q(p')$ holds for $p' < p$. We first discuss the case that p is even. Because

the property holds for $0, 1, \ldots, p-1$, there are exactly $\frac{p}{2}$ sweeps from sweep w through $w+p-1$ in which a data block is fetched for the users from G_{even} and for none of the users from G_{odd}. Likewise, we have that there are exactly $\frac{p}{2}$ sweeps in which a data block is fetched for each user from G_{odd}. Consequently, the time required for sweep w through $w+p-1$ is

$$\sum_{j=0}^{\frac{p}{2}-1}(\left\lceil\frac{n}{2}\right\rceil \cdot t_a(i_0+j)+s(\left\lceil\frac{n}{2}\right\rceil))+\sum_{j=0}^{\frac{p}{2}-1}(\left\lfloor\frac{n}{2}\right\rfloor \cdot t_a(i_0+j)+s(\left\lfloor\frac{n}{2}\right\rfloor)).$$

From (1.1), it follows that $s_2(n) = s(\lceil\frac{n}{2}\rceil)+s(\lfloor\frac{n}{2}\rfloor)$. This and rewriting yields that the previous expression equals

$$\frac{p}{2}\cdot(n\cdot t_d+s_2(n))+n\cdot\sum_{j=0}^{\frac{p}{2}-1}(t_a(i_0+j)-t_d).$$

Dividing by sb_{d}, which, by definition, equals $n\cdot t_d+s_2(n)$, gives

$$\frac{p}{2}+n\cdot\frac{\sum_{j=0}^{\frac{p}{2}-1}(t_a(i_0+j)-t_d)}{sb_{\text{d}}}. \tag{A.16}$$

which is the number of data blocks that is consumed during the first p sweeps. We will show by contradiction that the last fraction is at least 0. Hence, assume that the fraction is strictly smaller than 0. Because $2\leq p<2\cdot k_0$ we have

$$\sum_{j=0}^{k_0-1}(t_a(i_0+j)-t_d)=\sum_{j=0}^{\frac{p}{2}-1}(t_a(i_0+j)-t_d)+\sum_{j=\frac{p}{2}-1}^{k_0-1}(t_a(i_0+j)-t_d).$$

Because the first sum-quantification in the right-hand side is strictly less than 0, the second sum-fraction in the right hand-side of the equation has to be strictly larger than the left-hand side. This is in contradiction with the definitions of i_0 and k_0. Hence, we can remove the fraction from (A.16) without increasing its value. This yields that during the first p sweeps each user consumes at least $\frac{p}{2}$ data blocks. Moreover, as mentioned, exactly $\frac{p}{2}$ data blocks have been fetched for each user from the start of sweep w. Hence, after p sweeps the buffer of the users contain at most as much data as at the start of sweep w. Because, by assumption, the users from G_{even} have room for minimally one data block at the start of sweep w, this is also true at the start of sweep $w+p$. Furthermore, there has not been fetched a data block in sweep $w+p-1$ for any of these users, because by the induction hypothesis $Q(p-1)$ holds. Consequently, there will be fetched a data block for them in sweep $w+p$. This proves the induction hypothesis for the case that p is even.

We will now discuss the case that p is odd. Again, it follows from the induction hypothesis that there are exactly $\frac{p-1}{2}+1$ sweeps in which a data block is fetched for each user from G_{even} and $\frac{p-1}{2}$ sweeps in which a data block is fetched for each user from G_{odd} from sweep w through sweep $w+p-1$. Hence, the time required for sweep w through sweep $w+p-1$ equals

$$\sum_{j=0}^{\frac{p-1}{2}}(\left\lceil\frac{n}{2}\right\rceil \cdot t_a(i_0+j)+s(\left\lceil\frac{n}{2}\right\rceil))+\sum_{j=0}^{\frac{p-1}{2}-1}(\left\lfloor\frac{n}{2}\right\rfloor \cdot t_a(i_0+j)+s(\left\lfloor\frac{n}{2}\right\rfloor)).$$

Similarly as above we can derive that during this time

$$\frac{p-1}{2}+n\cdot\frac{\sum_{j=0}^{\frac{p-1}{2}-1}(t_a(i_0+j)-t_d)}{sb_{\text{d}}}+\frac{\lceil\frac{n}{2}\rceil\cdot t_a(i_0+j)+s(\lceil\frac{n}{2}\rceil)}{sb_{\text{d}}}$$

data blocks are consumed and that the second term is at least 0. The last term is clearly strictly larger than the upperbound of ε. Hence, from the start of sweep w the users of G_{odd} have consumed strictly more than $\frac{p-1}{2}+\varepsilon$ data blocks. Furthermore, exactly $\frac{p-1}{2}$ data blocks are retrieved, because, by the induction

hypothesis $Q(0)$ through $Q(p-1)$ hold. From the initial buffer content of the users from G_{odd}, it follows that they have room for at least one data block at the start of sweep $w+p$. Furthermore, no data block has been retrieved for any of these users in sweep $w+p-1$, because $Q(p-1)$ holds. This means that a data block is fetched for all users from G_{odd} in sweep $w+p$, which has to be proved.

We will use the property to show that buffer underflow can occur. From the property it follows that during sweep $w+2\cdot k_0-1$ the k_0th data block, counted from the start of sweep w, is fetched for each user from G_{odd} and that for each user of G_{even} exactly k_0 data blocks have been fetched from sweep w through sweep $w+2\cdot k_0-1$. It can be the case that the k_0th data block for a user from G_{odd} arrives at the end of sweep $w+2\cdot k_0-1$. From these observations it follows that the time between the start of sweep w and the arrival of the k_0th data block for a user from G_{odd} can be

$$\sum_{j=0}^{k_0-1}\left(\left\lceil\frac{n}{2}\right\rceil\cdot t_a(i_0+j)+s\left(\left\lceil\frac{n}{2}\right\rceil\right)\right)+\sum_{j=0}^{k_0-1}\left(\left\lfloor\frac{n}{2}\right\rfloor\cdot t_a(i_0+j)+s\left(\left\lfloor\frac{n}{2}\right\rfloor\right)\right).$$

As above, we can derive that during this time

$$k_0+n\cdot\frac{\sum_{j=0}^{k_0-1}(t_a(i_0+j)-t_d)}{sb_d} \tag{A.17}$$

data blocks are consumed. This number has to be at least the number of data blocks that an arbitrary user from G_{odd} has in the buffer at the start of sweep w plus the number of data blocks that are placed in the buffer from the start of sweep w. This number is given by

$$1+n\cdot\frac{\sum_{j=0}^{k_0-1}(t_a(i_0+j)-t_d)}{n\cdot t_d+s_2(n)}-x+\varepsilon+k_0-1.$$

Because, by assumption, $\varepsilon<x$ this expression is strictly smaller than Expression (A.17). Hence, buffer underflow can occur. □

Theorem 20. *If for all i, $0\le i<m$, (7.4) holds, filling strategy TB is used and the requests of the users are handled by Algorithm 4, then safeness is guaranteed if the buffers have room for $3-\delta_2$ data blocks, with δ_2 defined by (2.12).*

Proof. By the used filling strategy, no buffer overflow can occur. We have to prove that neither buffer underflow occurs. The outline of the prove is as follows. We define for an arbitrary video-on-demand system I, where multiple movies are stored on a disk that not necessarily cover the entire disk, a video-on-demand system I' that satisfies our assumptions, such that if buffer underflow occurs in I, then it also can occur in I', where in both systems filling strategy TB and Algorithm 4 are used and where in both systems the buffers can contain an equal number of data blocks that is at least $3-\delta_2$. By Theorems 3 and 15, safeness is guaranteed for I' if (2.10) holds. Consequently, if we prove that if (7.4) holds for I', then (2.10) holds for I, then safeness is also guaranteed for I.

Definition of I':
We let the movie of I' consist of $\sum_{i=0}^{m-1}n_B^i+m-1$ data blocks. Hence, the number of data blocks of the movie equals the sum of the data blocks of all movies from I plus $m-1$. We number the data blocks from 0 through n_B. Furthermore, we define the set of positions, i.e. $F_{I'}$, as $\tilde{F}_I\cup\{x_i|0\le i\le m-2\}$, where $x_i\notin F_I$, $0\le i\le m-2$, and $\tilde{F}_I$ is the set of positions from F_I on which a data block is stored. Consequently, $|L_{I'}|=|F_{I'}|$. We define $t_{I'}(i)$, $i\in F_{I'}$, as $t_I(i)$ if $i\in\tilde{F}_I$ and as 1 otherwise. All other values, such as the switch time function s, the block size B and the maximum number of admitted users n are chosen similarly to these values in I.

We define the assignment of the data blocks to the movie, such that its sequence is given by

$$seq(a^0)\mathbin{+\!\!+}x_0\mathbin{+\!\!+}seq(a^1)\mathbin{+\!\!+}x_1\mathbin{+\!\!+}\cdots\mathbin{+\!\!+}x^{m-2}\mathbin{+\!\!+}seq(a^{m-1}),$$

where $seq(a^i)$ is the sequence of a^i. Hence, the sequence of a' is the concatenation of the sequences of the

assignments of all movies separated by positions that are not in $\tilde{F}_I$. This assignment is clearly a bijection. Before we prove that if buffer underflow occurs in I, then buffer underflow can also occur in I', we introduce some definitions.

Introducing definitions:
We say that a data block in I equals a data block in I' if and only if it is stored on the same position. By the definition of the assignment, this means for example that watching the 0th movie in I is equivalent to watching the first n_B^0 data blocks from the movie in I'. From this, it follows that we can the compare the value of i_u in I with the value of i_u in I', except for the case that $i_u = n_B^j$ or $i_u = n_B$, which has the interpretation that user u does not want to watch a movie or that the last data block of a movie has already been fetched for user u. By definition, these values are equal, i.e., $n_B^j = n_B$, $0 \leq j < m$.

We define the *status of user* u at a given moment as the triple (f, req, i_u), where f is the number of data blocks in the buffer of u and req is the data block that user u has most recently requested and that has not been handled completely. If there is no non-handled request for user u, then $req = \perp$, by definition. The status of user u at the start of sweep w in instance J is denoted by $st_J(u,w)$. If the last data block of a movie is fetched, then the number of data blocks in the buffer is irrelevant for the status, i.e. if the last data block has both been fetched for user u and user u' in sweep w then $st_J(u,w+1) = st_J(u',w+1)$, regardless of the content of their buffer.

Buffer underflow in $I \Rightarrow$ buffer underflow in I':
We now prove by induction to the number of sweeps, denoted by w, that just before the start of sweep w the status of the users in I can also occur in I'. Hence, for each run in video-on-demand system I resulting in a given status for all users before the start of sweep w, there is a run of system I', such that $\forall_{1 \leq u \leq n}(st_I(u,w) = st_{I'}(u,w))$. Because the users have in both systems the same buffer size and because the same strategy is used in both system for scheduling disk accesses, this proves that whenever buffer underflow occurs in I, it also can occur in I'.

As basis we take $w = 1$. Hence, there has no sweep been executed. Initially, the buffers of all n users are empty. Consequently, letting the users initially request the same data blocks results in $st_I(u,0) = st_{I'}(u,0)$.

Consider the case that $w = v+1$. Assume that the status of each user in I at the start of sweep v equals the status of the same user in I' at the start of sweep v. By the induction hypothesis, for proving the induction step it suffices to prove that under this assumption this can also be the case at the start of sweep $v+1$. Because each user has the same status in I and I' at the start of sweep v, the same data blocks are fetched for the same users in sweep v.

Let each user consume at the same consumption rate during sweep v in both I and I'. Moreover, if u' sends a request in I, then the same request is sent by u' in I'. Let u be an arbitrary user. We have to prove that is possible that $st_I(u,v+1) = st_{I'}(u,v+1)$. We prove this by distinguishing several cases. By the assumption that when u sends a request in I, u sends the same request in I', we only have to discuss whether req is equal in both I and I' at the start of sweep $v+1$ if we let u send a request in I', while no request is sent by u in I.

Assume that no request is sent by u during sweep v and that no data block is fetched. Consequently, the value of i_u does neither change in I nor in I'. Moreover, the buffer contains as many data blocks in I as in I', because this is the case at the start of the sweep. Hence, if no data block is fetched for u because he/she has no room for it in the buffer, then $st_I(u,v+1) = st_{I'}(u,v+1)$. Otherwise, no data block is fetched because u is waiting for its request being granted. From Algorithm 4 it follows that the number of sweeps u has to wait before its request being granted only depends on the number of sweeps he/she is waiting and the last data block that has been fetched for u. Consequently, by the induction hypothesis, the request of u is granted between sweep v and $v+1$ in I if and only if this is the case in I'. Moreover, because $st_I(u,v) = st_{I'}(u,v)$, the same request is granted in I and I' if a request is granted. As a result, $st_I(u,v+1) = st_{I'}(u,v+1)$.

Assume that no request is sent by the user and that a data block other than the last data block is fetched

for the user, i.e., $i_u < n_B^j$, where j is the movie to which user u is watching. Then i_u is raised by one in both cases and a data block is places in both buffers. Note that if $i_u < n_B^j$ in I then also $i_u < n_B$ in I'. Consequently, $st_I(u, v+1) = st_{I'}(u, v+1)$.

Assume that no request is sent by the user and the last data block of the movie is fetched for u in I, i.e. $i_u = n_B^j$, $0 \leq j < m$. If $j = m-1$, then $i_u = n_B$ in I'. However, if $j < m-1$, then $i_u = x_j$. We then let user u send a request during sweep $w+1$ to stop watching the movie. By Algorithm 4, this request is granted immediately, which implies that $i_u = n_B$ at the start of the next sweep and that $req = \perp$ in both I and I'. Hence, $st_I(u, v+1) = st_{I'}(u, v+1)$.

Assume that a request is sent by u. Then both in I and I', the buffer of u is flushed. If $d(u) = n_B^j$ in I, then $d(u) = n_B$ in I'. Consequently, both requests are granted. Hence, $st_I(u, v+1) = st_{I'}(u, v+1)$. Otherwise, the user has to wait at least one sweep before the request being granted. Consequently, the value of i_u is equal in both I and I'. Which, again, means that the status is similar in both I and I'.

These case cover all possibilities. Hence, we have proved the induction hypothesis, i.e., we have proved that if buffer underflow occurs in I, then buffer underflow can also occur in I'. Finally, we have to prove that if (7.4) holds for I, then (2.10) holds for I'.

Equation (7.4) holds for $I \Rightarrow$ equation (2.10) holds for I':
By the definition of a, (2.10) holds for I', if $t_I \circ a^i(n_B^i - 1) + t_{I'}(x_i) \leq 2t_d$ and $t_{I'}(x_i) + t \circ a^i(0^{i+1}) \leq 2t_d$, $0 \leq i < m-1$. We prove the first equation. The second can be proved similarly. Let i be arbitrary. By assumption $n_B^i >> 1$. Consequently, data block $n_B^i - 2$ also exists. Because (7.4) holds, we have that $t_I \circ a^i(n_B^i - 2) + t_I \circ a^i(n_B^i - 2) \leq 2t_d$. By definition, $t_{I'}(x_i) = 1$ and by assumption $t_I(j) \geq 1$. Consequently, $t_{I'}(x_i) \leq t_I \circ a^i(n_B^i - 2)$. As a result, $t_I \circ a^i(n_B^i - 1) + t_I(x_i) \leq 2t_d$. This proves the theorem. □

Bibliography

[1] Y. Birk. Track-pairing: a novel data layout for vod storage servers with multi-zone recording disks. In *Proceedings international conference multimedia computing and systems*, May 1995.

[2] T. Chua, J. Li, B. Ooi, and K. Tan. Disk striping strategies for large video-on-demand servers. In *ACM MultiMedia*, November 1996.

[3] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[4] S. Ghandeharizadeh, D. Ierardi, D. Kim, and R. Zimmermann. Placement of data in multi-zone disk drives. In *Proceedings of the second international baltic workshop on DB and IS*, June 1996.

[5] S. Ghandeharizadeh, S. Kim, C. Shahabi, and Zimmermann R. Placement of continuous media in multi-zone disks. In *Multimedia information storage and management*. Kluwer Academic Publishers, 1996.

[6] S. Heltzer, J. Menon, and M. Mitoma. *Logical data tracks extending among a plurality of zones of physical tracks of one or more disk devices*. 1993.

[7] Y. Huang and S. Tsao. An efficient data placement and retrieval scheme of zone-disks to support interactive playout for video servers. In *IEEE transactions on consumer electronics, Vol. 43*, February 1997.

[8] J. Kim, H. Lim, Y. Kim, and K. Chung. A data placement strategy on MZR for VOD servers. In *Proceedings international conference on parallel and distributed systems*, December 1997.

[9] J. Korst. Random duplicated assignment: An alternative to striping in video servers. In *ACM Multi-Media*, June 1997.

[10] J. Korst and E. Lawerman. Comparing disk scheduling algorithms for video servers. Technical Report 307/96, Philips, 1996.

[11] J. Korst, V. Pronk, P. Coumans, G. van Doren, and E. Aarts. Comparing disk scheduling algorithms for vbr data streams. Submitted for publication in a special issue of *Computer Communications*.

[12] M. Lee, C. Wen, C. Cheng, and Y. Oyang. Designing a fully interactive video-on-demand server with a novel data placement and retrieval scheme. In *IEEE transaction on consumer electronics, Vol. 41*, February 1995.

[13] H. Lewis and C. Papadimitriou. *Elements of the theory of computation*. Prentice Hall, 1981.

[14] J. Mitchell, W. Pennebaker, C. Fogg, and D. LeGall. *MPEG video compression standard*. Chapman & Hall, 1996.

[15] S. Shim, H. Vedavyasa, and D. Du. An effective data placement scheme to serve popular video on demand. In *Proceedings of 3rd pacific workshop on distributed multimedia systems*, June 1996.

[16] Y. Wang, S. Tsao, R. Chang, M. Chen, J. Ho, and M. Ko. Fast data placement scheme for video server with zoned-disks. In *Conference proceedings SPIE, Vol. 3229*, November 1997.