UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

On Fault-Tolerant Data Placement in Storage Networks

Dissertation

by

Mario Mense

Faculty of Computer Science, Electrical Engineering and Mathematics
Department of Computer Science and Heinz Nixdorf Institute
University of Paderborn, Germany

August 2008

Reviewers:

- **Prof. Dr. Friedhelm Meyer auf der Heide, University of Paderborn, Germany**
- **Prof. Dr. Christian Schindelhauer, University of Freiburg, Germany**

Acknowledgments

First of all, I would like to thank my advisor Prof. Dr. Friedhelm Meyer auf der Heide for his great support. Friedhelm always left me the freedom to address the problems I was interested to investigate and which have now become the core of this thesis. Moreover, basing on his early results in PRAM simulation and resource management in networks, I had several motivating discussions with him from which I always could extract interesting proposals and useful perspectives for my work. Furthermore, I would also like to thank Christian Schindelhauer from Albert-Ludwigs-Universität, Freiburg, for reviewing this thesis. Collaboration with Christian is always interesting, progressive, sometimes strange, but in the end leads to fascinating views on many different problems.

At last, I express my thanks to those people with whom I had a great collaboration and great time, mainly the members of the research group "Algorithms and Complexity" and my co-authors of the main results in this thesis, namely Michael Kortenjan and Martin Ziegler from the University of Paderborn, Christian Scheideler from the Technische Universität München, Germany, as well as Christian Schindelhauer and Arne Vater from Albert-Ludwigs-Universität, Freiburg, Germany. Another special thank goes to Michael Kortenjan and Gunnar Schomaker for suffering me as their office member during the last years.

Paderborn, August 2008 *Mario Mense*

To Nicole and Charlotte

Contents

1 Introduction **3**

1.1 Storage (Area) Networks 4

1.1.1 The Storage Virtualization Layer 6

1.1.2 Storage Management Schemes 7

1.2 Demands on Data Placement in Storage Networks 9

1.2.1 A Brief Note on Erasure Encoding and Replication 12

1.3 Our Contribution 14

2 Some Classical Approaches **15**

3 Random Allocation of Data Copies **23**

3.1 Preliminary Section 26

3.1.1 The Basic Model 26

3.1.2 On the Practical Realization of the Allocation Function F 31

3.1.3 Further Preliminaries 39

3.2 The Redundant Allocation Problem 43

3.3 Analysis of the Probability Distribution P 46

3.3.1 Existence and Uniqueness 47

3.3.2 On the Imbalance Induced by Balls-into-Bins Allocations 49

3.4 COMB: An Allocation Strategy for Static Scenarios 60

3.5 SPREAD: An Adaptive Allocation Scheme for Dynamic Scenarios 64

3.5.1 Previous Adaptive Strategies 65

3.5.2 The SPREAD Strategy 69

3.6 Conclusion and Open Problems 89

4 Erasure Codes for Reading and Writing **91**

4.0.1 Preliminaries 93

4.0.2 Data Reconstruction from Failures 96

4.1 The Operational Model 101
4.2 Lower Bounds 103
4.3 Encoding/Decoding in the RWC 104
4.3.1 The Matrix Approach 106
4.4 Security and Redundancy 111
4.5 Adaptive RW Codes 112
4.6 Boolean Read-Write-Codes 114
4.7 General RW-Codes 118
4.8 Conclusion 120

CHAPTER 1

Introduction

The advances in networking technology over the last decades, the explosive popularity of the World Wide Web and, depending on this, the growth of the Internet have greatly widened the user base of computers, making it nearly universal. In times of the so-called *Web 2.0*, information exchange has become ubiquitous, and novel pervasive applications, like e.g. online banking, audio- and video downloads, data warehousing or the wide and continuously growing field of e-commerce, have been enabled by this progress. Furthermore, it has become usual for people in the industrial nations that information is available 24 hours a day, 7 days a week[1], and complex multimedia based content, like video clips and MP3 music files, can be downloaded at high rates or viewed online.

Driven by this evolution, data has become the central and most valuable asset for many companies and organizations, and generally, the efficient storage and retrieval of huge amounts of information has emerged as a driving force of our information society's economy. Due to an ICD study sponsored by EMC[2] [Gan07], the worldwide amount of digital data in 2006 is projected at 161 Exabyte (billion GByte). In other words, on average, each of the 6.7 billion people on earth has created 24 GByte of new data in 2006. This is plausible since multimedia applications have entered a phase of rapid growth which is driven by mainly two forces. First, the rapidly dropping costs of computer and networking hardware has made multimedia applications accessible to an ever-increasing number of users. Second, the growing size of the Internet has made it possible to publish multimedia content comparatively cheaply to an immense audience. For example, YouTube, a company that did not exist a few years ago, hosts 100 million video streams a day, and experts say more than a billion songs a day are shared over the Internet in MP3 format. Assuming an expansion rate of 57%, in 2010, the worldwide data volume will be about six times as much as today, 988 Exabyte. Particularly, from the total global data volume, about 70% of all information is created by private persons, but nevertheless, roughly 85% of all data

[1] often abbreviated 24/7 or 24x7

require the involvement of a company in terms of storing, providing or transmitting the data. As a result, for providing and enabling all this information, multimedia content and technical processes, companies have to administrate, process and backup these quantities of data appropriately, and it has turned out that, on average, enterprises have to deal with an information amount that doubles every year.

Therefore, in the recent years, stored data has become more and more critical and irreplaceable, and the demand for quick and uninterrupted data availability has reached highest priority, thus, basic issues such as fault-tolerance, continuous information availability, fast data access and advanced backup principles are crucial for every company today. To guarantee all this, the concepts to manage and store data are of great importance because the quality of the data availability as well as the access speed have direct impact on any company's success. As an example, consider the case of a video-on-demand service provider storing all his frequently accessed data on the same storage device. Due to bandwidth limitations, during playback, the device may become unable to satisfy all incoming request appropriately leading to loss of customer goodwill and hence, may be considered expensive. As a result from summarizing all demands addressing modern advanced data allocation, we can identify two major objectives any contemporary storage subsystem has to accomplish: high-level *serviceability*, i.e. always serve given I/O requests quickly and efficiently, independent of either the network's size or occurring changes in the set of storage devices, and *data availability (or reliability)*, i.e. to feature appropriate fail-over mechanisms to be safe from disk failures, and thus, data loss.

1.1 Storage (Area) Networks

The technological foundation to face the dramatic growth of enterprise data storage capacity as well as to meet the given objectives is to consolidate the storage capacity inside a so-called *storage area network (SAN)* (see e.g. [JT05, Gup02] for details), a dedicated network which is built around block-addressed storage units and that disbands the traditional tight coupling of servers and storage devices, but instead, establishes an any-to-any connection from servers to disks by an underlying high-speed network. This concept of *storage consolidation* leads to the so-called *storage-centralized architecture* enabling enterprises to facilitate efficient parallel access to the data as well as cost-effective system management. Unlike common server-centralized architectures in which a single disk drive is directly attached to a workstation, the disks are rather encapsulated inside advanced storage subsystems, like e.g. disk arrays or high-end enterprise storage cabinets, that today offer a capacity ranging from some TByte to several PByte each. Every such storage subsystem can be directly accessed by any of the connected servers, thus, enhancing the ability to assign free capacity in an easy and flexible manner (see Figure 1.1). Additionally today, almost all SANs are scalable although not all implement convenient

storage-on-demand concepts, that is, to add (theoretically unlimited) storage capacity to the system only when it is needed. Another problem that comes along with scalable environments is *heterogeneity*. Depending on the underlying storage configuration (c.f. Section 1.1.2), the ability of a storage network to scale may induce the storage components to become heterogeneous over time because new storage devices enter the system as well as older ones are being removed or replaced. This indeed simplifies migration processes, but on the other hand forces the management modules to be able to cope with heterogeneity. However, even if not all SANs are capable of handling heterogeneity yet, the ability to scale already reduces the total costs of ownership (TCO) [GM00, Fal07, BSV04].

Last but not least, every SAN is required to have appropriate reliability mechanisms installed to face the mentioned problem of data loss in case of device failures, while the probability of such failures to occur increases with the number of storage devices in the SAN (c.f. [PGK88] and Section 1.2). The common way to achieve high data availability, and moreover, to ensure *business continuity* appropriately, is given by inducing redundancy into the system in terms of applying either complete replication of data blocks or erasure (resilient) encoding schemes (e.g. RAID). Unfortunately, almost all utilized fault-tolerant placement strategies suffer from serious drawbacks, like e.g. the lacking ability to properly cope with heterogeneity as well as insufficient failure resilience or inherent I/O overheads when modifying the data.

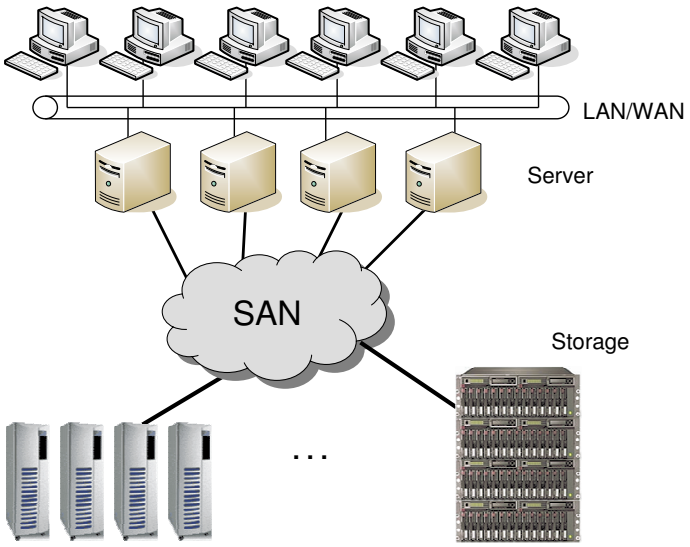

Figure 1.1: Storage Area Network

To summarize, depending on storage consolidation and supported by advanced data and resource management strategies, modern SANs offer a high potential regarding the efficient adaption to changing capacity demands without system downtimes as well as parallel access to the (non-uniform) storage devices, low maintenance costs, simplified

administration and integrated recovery services to react on hardware failures. Surely, to ensure all these benefits, appropriate mechanisms to provide *fault-tolerance*, *system scalability* and *efficient data access* even for non-uniform storage devices must be installed.

1.1.1 The Storage Virtualization Layer

Inside a SAN, the utilized data placement scheme is the building block to face the sketched challenges and which is usually part of a software layer implementing the so-called *storage virtualization*, an abstract concept separating the physical from the logical view of storage resources (see e.g. [BMS+03, BHMadH+04, Sal04]). This abstraction enables the consolidated storage devices to be managed as one single entity, normally as one virtual disk, which is then offered to the application servers while hiding the physical representation and explicit storage of the data from the user (Figure 1.2). In particular, the logical address space presented to the accessing servers in terms of virtual volumes is scrambled into equal sized data blocks which are then distributed among the physical devices by the data distribution strategy according to the desired serviceability and availability objectives (additionally, modern SANs offer improved management services, like e.g. remote copy or volume snapshots [BEHV06]).

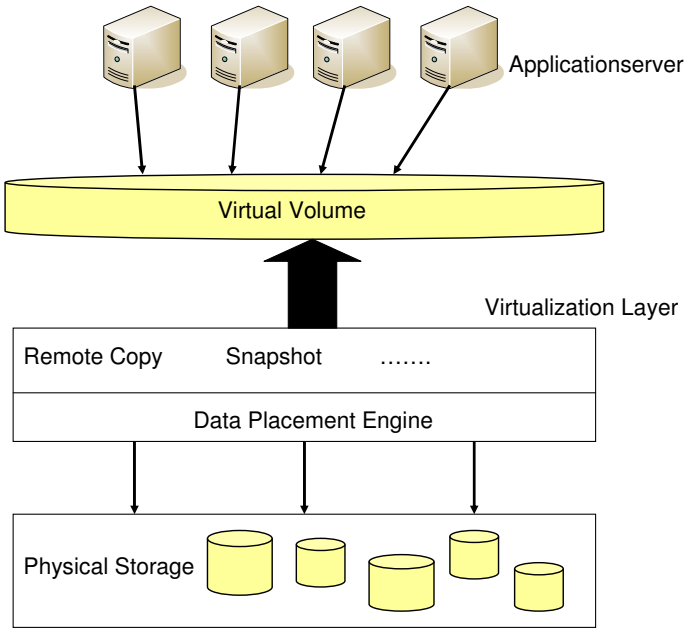

Figure 1.2: Storage Virtualization Layer

However, obviously, both the given storage management scheme (see next section) and, depending on the layout, the capability of the utilized allocation scheme directly determines the overall performance of the storage network regarding the described challenges.

1.1.2 Storage Management Schemes

In most commercial SANs today, RAID is still the usual way to allocate data blocks to the set of storage devices, and for fault-tolerance purpose, most companies apply RAID levels 1 (replication, resp. mirroring) or 4/5 (erasure (resilient) encoding) (for details, see [PGK88, Mas97]) which arranges the scrambled data blocks into a fix sized stripe of data blocks and additionally includes a parity block for providing one-failure tolerant redundancy; if a higher level of redundancy is required, often *Reed-Solomon erasure encoding* is considered (c.f. Chapter 4). Although RAID implements a deterministic (resp. manual) data placement, it works for small storage environments as RAID sets are shipped in terms of rigid storage cabinets each covering some fixed number of homogeneous disks on which the data is striped by an internal controller to ensure low system latencies. Surely, fixed RAID sets provide optimal fairness (by uniform data distribution) and since such systems can tolerate one single disk failure at a time, vendors almost add so-called *spare disks* for compensating a missing disk in case of failure.

However, if considering large-scale systems or those that may undergo changes in the number of connected storage systems, the traditional rigid layout of such SANs makes storage management become more and more less efficient because the only way to scale is to concatenate multiple RAID sets disjointly, as depicted in Figure 1.3.

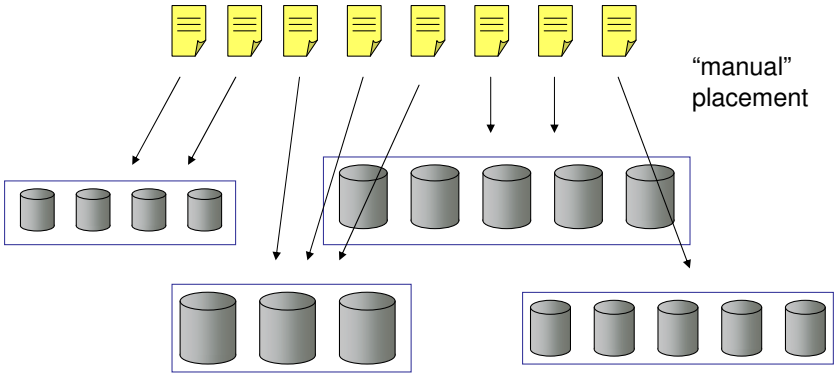

Figure 1.3: Classical Storage Management: Concatenated RAID sets

The drawbacks are quite obvious. First, if such systems base on disks of same type, the probability of an occurring failure increases significantly (c.f. next section). Furthermore, easy system and data migration is merely possible inside a given cabinet, but most importantly, since the storage virtualization is located close to the data inside the cabinets, the benefits of virtualization, like e.g. fault-tolerance, disk utilization or access efficiency, have only local effects inducing the need for an additional superior virtualization layer. If then such layer also implements RAID, system scaling becomes a hard challenge because if new disks are added over time, with high probability, they are of different characteristic making the scaling problem a heterogeneous one.

Therefore, a self-evident way to achieve highest system performance is given if environments get rid of all the rigid cabinets and arrange the complete storage in a single, flat storage solution, as depicted in Figure 1.4 ? Surely, such solution is much more challenging to the employed allocation scheme because all mentioned issues directly address the superior virtualization layer. However, besides the possibility to efficiently handle heterogeneous devices, an open, flat storage environment further offers increased scaling possibilities as well as better capacity consumption by efficiently using the complete storage space and thus, letting spare disks become obsolete.

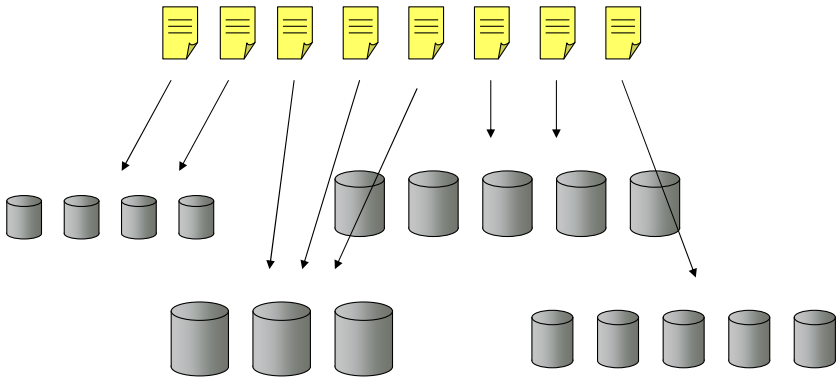

Figure 1.4: Open Storage Management: Single, flat storage arrangement

In this work, we present and discuss different data placement schemes for both *classical* and *open storage configurations*, and thus, depending on different restrictions, also different demands with respect to the virtualization engine are considered. For example, for a classical RAID-based storage layout as described, we ignore scalability and heterogeneity, and instead, rather focus on fault-tolerance and efficient access behavior that is almost gained by erasure resilient encoding.

Therefore, in Chapter 4, we present a flexible framework for generating different erasure resilient codes that, compared to classic erasure codes, include improved update behaviors which are very useful for SANs. On the other hand, in case of open storage environments, scalability and heterogeneity additionally come into play forcing the applied distribution scheme to handle all challenges mentioned in the previous section. Therefore, in Chapter 3, we introduce efficient and redundant strategies that base on replication and which are furthermore capable of coping with scalable and heterogeneous storage configurations, respectively. However, before we go into further detail investigating these novel allocation schemes, we first introduce and briefly describe the complete set of different requirements that generally address data allocation and which must, at least partially, be handled by any placement algorithm employed in a storage network.

1.2 Demands on Data Placement in Storage Networks

We give a brief summary of the basic challenges that must at least in part be considered by a data allocation scheme to ensure the serviceability and availability objective for a considered storage environment. In particular, as the underlying storage configuration may restrict the set of demands to be considered by the applied placement scheme, different overall performances might be achieved for varying configurations. However, all strategies observed in this thesis in common is that they can be reduced to pure data placement only, i.e. all considered strategies are not concerned with limiting properties of the underlying network topology, like e.g. congestion or network latency. This results from the fact that the I/O accesses to the single storage devices cause the bottleneck of the overall system because any such access is served by comparatively slow mechanical driven access mechanisms inside a single storage device. Consequently, we can suppose the underlying network topology to be unrestricted because the system bottleneck appears directly at the storage devices such that potential problems when routing the data packets through the network can be neglected.

Therefore, the usual way to ease the access bottleneck at the disks in order to decrease system latency in both the classic and the open storage model is *data striping* which spreads the data blocks of some given file to multiple storage devices making multiple portions of the file be accessible in parallel, thus yielding effective transfer rates. Basically, the principles of data striping can be partitioned into random and deterministic approaches, and as we have seen, in traditional (commercial) SANs, respectively disk arrays, the well-known deterministic *RAID* scheme [PGK88, Mas97] is almost still applied. However, in general, besides parallel access, striping the data as evenly as possible among the disks further results in an optimal disk utilization which, at first, also improves system performance because each write I/O involves several disks for data storage, and second, reduces the TCO (c.f. [Mas97]). Moreover, data striping over multiple devices considerably supports *business continuity* which, for instance, becomes apparent in the example of the video-on-demand provider who runs into problems when storing frequently accessed data onto either a single or a small number of devices such that parallelism could not be exploited accurately. On the other hand, placing only infrequent accessed material on a storage device may result in the device to be underutilized. Thus importantly, any allocation scheme must provide an appropriate *load balancing* with respect to both the data blocks and the incoming I/O requests over all accessible disks.

As we explained already, modern open scalable storage systems additionally need to consider storage devices of non-uniform characteristics (e.g. capacity, bandwidth, latency, age etc.). Therefore, advanced data distribution strategies are required to handle heterogeneity appropriately; with respect to Salzwedel [Sal04], we denote this the *heterogeneity problem*, and any data distribution strategy able to efficiently handle heterogeneity

is called *fair*. Similar to uniform capacities, a fair distribution has the same great influence on disk utilization and thus, overall system performance.

Another issue significantly affecting system latency is *system scaling* what is caused by an ever growing storage data volume as well as occurring disk failures. In case of a growing data volume, the simple concatenation of storage devices obviously implies access bottlenecks caused by the so-called *80/20 problem* describing the fact that without data redistribution, new data to which roughly 80% of all current accesses go is almost exclusively stored on the newer devices covering roughly 20% of the total storage capacity (c.f. Chapter 2). Additionally, depending on technical progress, age reasons or failure probability, added or replaced disks are often of different characteristic compared to the ones already connected. Thus, even if initially of homogeneous trait, over time, a SAN becomes more and more heterogeneous. Furthermore, to obtain the best performance, the newly added disks should also be included into the striping process but this evokes the problem of how to add newer disks efficiently such that, in the long run, the overall performance is not affected noticeably. Clearly, in case of fixed sized data stripes, any system growth induces a complete redistribution of almost all of the stored data. If then TBytes or PBytes of data are stored, either serious performance losings or complete downtimes of the system for hours or days must be considered, what is intolerable for every company. Thus consequently, system expansion should be done as fast as possible, and after finishing the redistribution, the strategy should *balance* both the data and the requests optimally over all disks. A scalable data placement scheme that keeps the amount of redistributed data at a near-minimum is called *adaptive*.

Besides performance considerations, the stripe size has further influence on the reliability of the system because, according to Patterson et al. [PGK88], the failure rate of a disk array increases proportional to the number of covered disks which are expected to fail independently. More precisely, failures in disk arrays are often assumed to satisfy the memoryless property, that is, the life expectancy of a disk is estimated only upon the condition that the disk is working now. Hence, the reliability of a disk array is modeled by the exponential distribution [Gib99]. Consequently, for low disk failure rates, the failure rate of a disk array is proportional to the number of disks it contains. However, as a rule of thumb, we can alternatively consider the so-called *Mean Time Between Failure (MTBF)*, the vendor given operation time of a single disk expressing its reliability (for example, the Hitatchi Ultrastar^{TM}A7K1000, a conventional disk in 2007, exhibits an MTBF of targeted 1.2 million hours [Tec07]). Assuming a constant failure rate, the MTBF of a disk array then is

$$\text{MTBF of a Disk Array} = \frac{\text{MTBF of a single disk}}{\text{Number of disk in the array}}$$

Thus, for a SAN consisting of 100 Ultrastar^{TM}disks, the MTBF is calculated as roughly 1.2 million / 100 hours (= 1.5 years). At a first glance, this seems sufficient, but if we

scale up to 1000 disks, the MTBF drops to 1,200 hours, or less than 2 months, what is dismal. With respect to the disk failures that might occur, we primarily distinguish three types. The first, called *transient errors*, arise from noise corruption and are dealt with by repeating the requests. The second, called *media defects*, are caused by permanent defects in material and are detected and masked by the manufacturer. The last, on which we concentrate in this thesis, are *catastrophic failures* such as head crashes and failures of the disk controller electronics. According to Hellerstein et al. [HGK+94], we denote such failures *erasures* throughout this work because when a disk suffers a catastrophic failure, its data is rendered unreadable, and thus, is effectively erased.

Consequently, besides appropriate backup mechanisms, any allocation scheme should furthermore provide some certain degree of fault-tolerance. Usually, fault-tolerance is achieved by introducing redundancy into the system that can be realized by either *replication*, i.e. seperately storing identical copies of every data block, or *erasure resilient encoding* by which the information is redundantly encoded inside a codeword made up of a number of blocks that are stored on separated devices. As we see in the following chapters, erasure encoding is suitable rather for a homogeneous set of disks whereas with replication, heterogeneity and scalability can be tackled efficiently. That is, all data placement strategies we introduce in this work are fault-tolerant approaches while redundancy is either provided by replication or erasure resilient encoding. However, the central algorithmic problem we address in this thesis is the following

Data Availability Problem: *Guarantee to always ensure a fast and efficient access to all stored data in a storage network even in case of erasures.*

Surely, the data availability problem covers the fulfillment of both the serviceability and the availability objective defined in the beginning including all properties and requirements discussed above. In the following list, we summarize these general requirements that result from the given discussion:

1. **Redundancy:** Data items are redundantly stored on separated disks.

2. **Fairness:** To ensure a best optimal disk utilization and reduced access latency, a data placement strategy must assign every disk a fair share of the total data load, i.e. a share of data that is equivalent to its share of the total capacity. This should hold not only for the data, but also for the I/O requests.

3. **Heterogeneity:** Non-uniform capacities or bandwidths should also be handled optimally.

4. **Scalability:** The system should be able to efficiently handle the addition or removal of storage devices.

5. **Adaptivity:** A scheme is called *adaptive* if in case of any system changes (in the number of data items or storage devices), it allows to adapt to the new configuration with minimum amount of data replacement while maintaining all given demands.

6. **Availability:** Suitable redundancy mechanisms should be incorporated to compensate the loss of failed (or blocked) disk drives.

7. **Efficiency:** The total space for storing control information should only depend on the number of storage devices and not on their differences in capacity. Furthermore, the time for computing the position of any data item stored in the system should be at most logarithmic in the system size.

Again, driven by specific application demands concerning the applied allocation strategy as well as different characteristics of the given hardware environment often allow a placement scheme to cover only a subset of the listed issues, as we see in the following chapters.

Before we investigate the different strategies in more detail, for the sake of completeness, we first take a brief look on the parameters that must be chosen for replication as well as erasure coding because in large-scaled storage environments, a proper selection of values for the redundancy parameters directly influences the capacity provisioning and thus, the TCO of the company.

1.2.1 A Brief Note on Erasure Encoding and Replication

Replication is a simple scheme where r identical copies of each data object are stored separately on r different disks, thus, enabling the system to tolerate up to $r-1$ erasures. In contrast, with an erasure-coded redundancy scheme, each object is divided into k fragments and redundantly encoded into a codeword of $n > k$ fragments (blocks) which are also stored separately. Then, the key property is that the original information can be reconstructed from any k fragments of the encoding. Basically, the main difference between replication and erasure encoding is that with replication, the entire information of the data block is stored in each copy implying that only one out of r copies suffices for recovering the data, what is an advantage as most applications today feature a rather read-intensive access behavior. On the other hand, this leads to a serious drawback if capacity savings become apparent as replication suffers from a storage overhead of a factor of $r-1$. Therefore, the main reason for using erasure coding (if only homogeneous storage devices are considered) is that a high degree of redundancy, respectively data availability, can be achieved with less additional storage because the redundancy factor is only $s_c = \frac{n}{k}$. Nevertheless, in case of non-uniform capacities or a scalable storage system, replication features the discussed advantages.

Often, the values r, k and n are simply set by a rule of thumb but in large systems, made up of a hundred or thousands of storage devices, these values should rather be chosen carefully with respect to the mentioned storage overhead. Moreover, this also includes the question of how many copies to hold for data items featuring different popularities. For this problem, a suitable way to determine and thus control the storage overhead of both approaches was presented by Rodruiges and Liskov [RL05]. They focus on high availability in peer-to-peer networks and model the values of r, respectively s_c, according to a desired per object unavailability target $\varepsilon > 0$ in combination with an average node availability a denoting the time a peer is reachable. In particular, in case of replication, they assume the node availability to be distributed independently and identically. Then, the values for ε and r, respectively, are computed as

$$\begin{aligned}\varepsilon &= \Pr[\text{object } o \text{ is unavailable}] \\ &= \Pr[\text{all } r \text{ replicas of } o \text{ are unavailable}] \\ &= \Pr[\text{one replica is unavailable}]^r \\ &= (1-a)^r\end{aligned}$$

which upon solving for r yields $r = \left\lceil \frac{\log \varepsilon}{\log(1-a)} \right\rceil$.

For the case of erasure coding, we sketch a summary of the complete derivation to compute s_c which can be found in [BSV03]. Object availability is given by the probability that at least k out of $s_c \cdot k$ fragments are available which can be modeled by a binomial distribution as

$$1 - \varepsilon = \sum_{i=k}^{s_c k} \binom{s_c k}{i} a^i (1-a)^{s_c k - i}.$$

Applying some algebraic simplifications and the normal approximation to the binomial distribution, the storage overhead s_c can be obtained as

$$s_c = \left(\frac{\sigma_\varepsilon \sqrt{\frac{a(1-a)}{k}} + \sqrt{\frac{\sigma_\varepsilon^2 a(1-a)}{k}} + 4a}{2a} \right)^2$$

where σ_ε is the number of the standard deviations in a normal distribution for the required level of availability.

Besides the result of Rodrigues and Liskov [RL05], there exists further comparisons [BSV03, DLS+04a, WK02], all of which argue that erasure coding is the clear victor due to capacity savings for the same level of availability. However, the authors only compared approaches for homogeneous, unscaled systems, and as we already described, in case of non-uniform capacities or dynamic system behavior, an open replication based allocation has considerable advantages.

1.3 Our Contribution

Depending on the different storage configurations briefly described in Section 1.1.2, the schemes we introduce in the following are coarsely divided into two different parts. With respect to an open, heterogeneous and perhaps dynamic system, in the first chapter, we investigate replication-based schemes that adapt balls-into-bins games (see Section 3.3.2.1) for randomly allocating identical copies of data items to the set of storage devices ensuring that no two identical copies are hosted by the same device. Thus, given a replication parameter of r, the system can tolerate up to $r-1$ failures and is still operational. We consider randomized algorithms because in the non-redundant case, a random allocation of the data items has turned out to be a useful way to conveniently model the load balancing problem. Furthermore, we distinguish between static (i.e. not dynamic) and dynamic environments that may undergo changes in the number of connected disks, and as we see, if scalability is neglected, a random and fair allocation to non-uniform capacities can already be achieved very easy. Nevertheless, as we show, with a random and redundant allocation of identical copies to non-uniform disks, an appropriate load balancing is harder to gain than in the non-fault-tolerant case. Furthermore, if the system is allowed to behave somehow dynamic, this challenge increases by magnitudes, but as we show by the SPREAD strategy, an adaptive allocation scheme for redundant and fair storage in dynamic heterogeneous storage systems, this can be handled efficiently.

Although storing replicated copies implies the efficient and appropriate handling of hardware heterogeneity as well as dynamic behavior, unfortunately, such schemes require to update all copies in case of any data modification to keep the system consistent, and moreover, they require a higher waste of resources. Therefore, in the second part, we introduce the so-called *Read-Write Erasure Coding Systems (RWC)*, a flexible system for generating parametrized erasure codes that, unlike usual erasure codes such as RAID or Reed-Solomon codes, offer improved update properties making them very useful for utilization in storage networks. Furthermore, the generated codes additionally include new security features for a secure storage of redundant data.

Before we start introducing our main results, we first sketch some of the most prominent previous results, each of which was designed to tackle a certain subset of the listed issues above.

CHAPTER 2

Some Classical Approaches

In this chapter, we give a brief survey of the most prominent approaches in the field of data placement algorithms that are usable for storage networks. Basically, most of the schemes we discuss in this thesis distribute the given data by applying either hashing methods, randomization, erasure resilient encoding or a mixture of these techniques, and for most of them, it holds that depending on given parameters, configurations or objectives, it often suffices to fulfill only some of the number of listed demands on data placement. For example, if the system is not expected to scale and the connected storage devices are of uniform size, simple and efficient XOR-based RAID schemes could be applied, instead of complex distributed hash table approaches.

Since we investigate randomized allocation schemes that adapt balls-into-bins games and erasure resilient encoding more deeply in the two major parts of this thesis, we now discuss alternative approaches that base on a different design. All these approaches in common is that non of them covers the data availability problem completely, thus, we partition the description of the algorithms according to the requirement they mainly aim to fulfill, namely fairness, data availability, heterogeneity and scalability (moreover, in this chapter, we restrict ourselves to scalability rather than adaptivity because for some scalable strategies the degree of adaptivity has not yet been analyzed or is simply ignored).

As fairness is one of the most important issues with respect to the serviceability objective, we start with introducing techniques that aim at both obtaining a fair data and access distribution over the disks to provide optimal system response times. In particular, these schemes can mainly be distinguished into either deterministic or random based ones.

Fairness

Again, the ability to distribute the data in a fair manner among the (perhaps non-uniform) disks in a SAN (which is often also referred to as *space balance* [Sal04]) directly correlates to an optimal system saturation, respectively an overall disk utilization. Moreover,

besides decreasing the waste of storage space by full capacity usage, inside a disk, continuous access algorithms, like the elevator strategy, are applied making the response time of a disk correlate with its utilization.

Moreover, as briefly sketched in Section 1.2 and shown by the example of the audio- and video provider in the previous chapter, not only the data must be distributed fairly; the second factor that has great influence on the performance are *user access patterns*. In particular, it has turned out that the frequency of access to data objects follows a *Zipf distribution*[1] [WAS+96] (Figure 2.1) where about 80% of the accesses are to 26% of the data indicating a high locality of reference which leads to access hot spots that have to be dissolved by the placement algorithm. Unfortunately, many of the older expendable data placement schemes do not consider this problem appropriately. For example, Fargin et al. introduce *Extensible Hashing*, a scheme that uses a directory for addressing pages [FNPS79, ED88]. The scheme applies a hash function to evenly spread the data across the directory, and the addresses of the pages are stored in at least one index of the directory. An important characteristic of this directory is that several entries may point to the same primary data page which leads to access clustering on the pages. Moreover, in case of key overflows on some pages, the directory expands by complete doubling. In that case, another hash function addressing a wider index range is applied which in turn intensifies the access clustering on some pages.

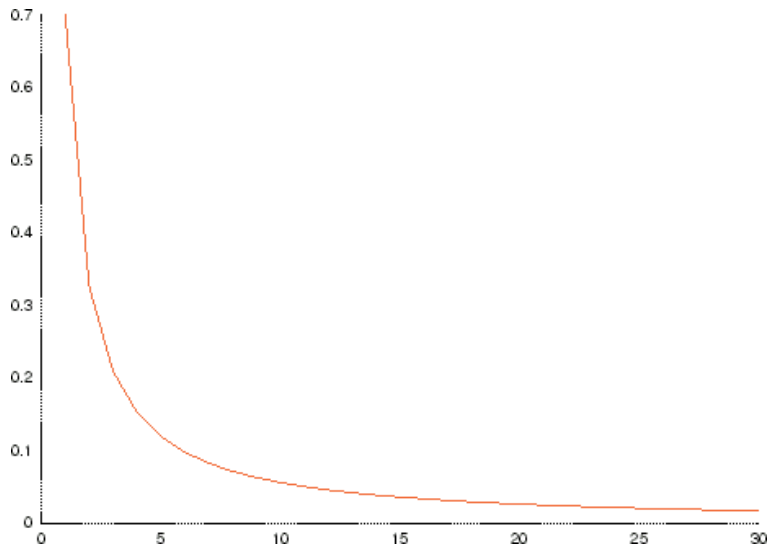

Figure 2.1: A typical Zipf-law rank distribution. The y-axis represents occurrence frequency, and the x-axis represents rank (highest at the left).

[1] In a Zipf distribution, if the objects are sorted due to the access frequency, then the access frequency for the ith object is given by $p_i = c/i^{(1-\theta)}$, where θ is the parameter for the distribution and c is a normalization constant.

Therefore, space and access balance are mutually dependent, and it is always necessary to store the data such that I/O accesses are balanced evenly across the disks, regardless of using uniform or non-uniform sized disks.

When looking at algorithmic approaches, usually in storage networks, deterministic strategies are used, while the most-applied deterministic technique is *uniform striping* [Joh84, PGK88, Mas97, HJH02] by which, given a fixed stripe size n, the i-th element in the data stripe is mapped to disk $D_{i \bmod n}$ on position i div n (c.f. the widely used RAID schemes with levels 0/4/5/6/...). However, deterministic striping is hard to scale because whenever a fixed stripe size is broken up and scaled to a different size, this induces data redistributions at great expense. Therefore, the natural extension is to simply concatenate additional disk arrays to the given system, but again, this leads to the above 80/20-problem (also discussed in the previous chapter). An alternative approach is to group the disks into clusters in advance while each cluster i defines its own stripe of length ℓ_i such that inside the cluster, uniform striping is applied. Unfortunately then, either the complete system has to be planned also for future situations or, in case of concatenated disk arrays, the number of new disks is determined by fixed stripe sizes perhaps resulting in an undesired over-provisioning of storage capacity. As a consequence, there exists a number of simple striping variations to break the very regular structure of this approach (e.g. [BGMJ94]).

A more general drawback of deterministic schemes is given by the fact that if the placement policy is completely known, bad inputs causing placement collisions can easily be created. Thus, the most effective protection to be safe from bad input is given by randomization (see e.g. [CW77, LYD71, AT97, SMRN00] and c.f. Section 3.1.2.1). With a randomized placement scheme, each data block is assigned to a disk drawn at random by either a pseudo-random hash function or a precomputed random distribution [SM98a]. Generally, randomized placement schemes are often much simpler than their deterministic counterparts, efficient to use, redundantize the need for complex and sensitive centralized control, provide efficient results even for non-uniform settings, and most importantly, unlike deterministic striping patterns, offer high potential to adaptivity, i.e. being able to react quickly on occurring configuration changes (see [Sch00] for a good overview about randomization techniques and Chapter 3 for efficient novel strategies).

Scalability

Again, todays SANs still follow the classical storage management (Section 1.1.2) for which the expansion rule is simple concatenation of rigid RAID arrays in case of system scaling. As mentioned above, this results in an undesired and costly capacity over-provisioning implying a negative effect on the TCO. Moreover, assuming a fixed stripe size, the induced redistribution phase dramatically influences the overall system performance.

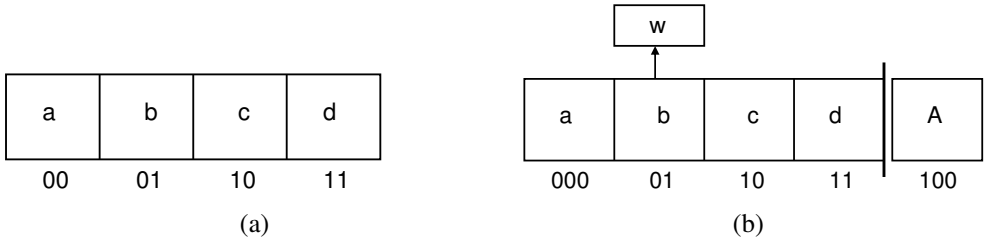

Figure 2.2: Linear Hashing

An early approach on scalability to overcome these drawbacks is given by the *Linear Hashing* strategy providing a fair and access balanced distribution of data records directly on a number of continuous (primary) buckets as long as the address space remains unchanged. The buckets are addressed by a pair of hash functions h_i and h_{i+1}, $i = 0, 1, 2, \ldots$, starting with h_0, and each h_i hashes keys to primary buckets (Figure 2.2a).

As shown in Figure 2.2(b), continuous key insertions on single buckets may lead to bucket overflows (bucket *b*) inducing the address space to expand which is done by splitting overflowed buckets in two starting from left to right (that is, bucket *a* is split before bucket *b*) while one of them is added as a new (secondary) bucket at the rightmost end of the primary address space. To address the split buckets, a linear extension h_{i+1} of h_i is then used for addressing the new (extended) address space.

However, although designed for scaling systems, Linear Hashing features serious drawbacks. First, the continuous (and linear) addition of buckets leads to bad space utilization as unsplit buckets are more likely to overflow than newly added ones leading to waste of storage. Second, the best performance for an unscaling system is gained by hash functions that distribute the data uniformly over the address space. Unfortunately, as new buckets are sequentially added at the rightmost end, the overflow chains of buckets at the rightmost end become too long (Figure 2.2) because they are the last to split, thus, causing an imbalance. Therefore, a decreasing exponential distribution that maps more keys to the left end would perform better. However, there are different approaches that handle this problem accurately but by inducing slightly more complexity into the system [Mar79, Mul84, Oto88].

Basing on the idea of Linear Hashing, Litwin et. al introduced *LH** [LNS93], a class of scalable distributed data structures (SDDS) generalizing the concept of Linear Hashing to parallel and distributed RAM and disk files without employing a centralized coordinator. In contrast to Linear Hashing, the address space is shared and manipulated by several clients, and to enable distributed processing appropriately, all clients have their own view on the file, and calculations are made on local values. Accurate object manipulations are then coordinated by a messaging system where clients process local parameter adjustments while only the bucket splitting is controlled by a dedicated server, the split

coordinator.

Like Linear Hashing, LH* allows a file (address space) to grow gracefully in a distributed environment but adopts many of the drawbacks sketched above. As an extension, several LH* variants have been created that incorporate fault-tolerant features, such as mirroring, checksum, scalable availability or Reed-Solomon encoding (see e.g. [LN96, LMR98, LS00, LR02]). However, depending on the overflow policy, space utilization is also more or less moderate, and more importantly, no LH* variant provides mechanisms to cope with non-uniform disk capacities.

Data Availability

We already explained that the easiest way to achieve fault-tolerance is to separately store identical copies of all data items on different disks, like in the RAID 1 scheme [PGK88] or the *PAST storage server* [RD01, DR01], and although keeping additional copies for each data item results in a higher waste of storage capacity, compared to erasure resilient encoding, replication offers better access properties for read-I/O dominated applications, like not being concerned with complex data en- or decoding as well as offering the chance for accesses to alternate on identical copies for I/O speed-up. Nevertheless, instead of replication, commercial SANs rather use RAID-parity encoding while business continuity is provided by the concept of *(hot) spare disks*, dedicated disks that are kept for fail-over only. Negatively, in normal mode, these disks remain unused, and only in case of a failed disk, they take over the missing part. Additionally, this concept does not ensure the expandability of the system due to growing storage demands.

With respect to the LH* schemes, Honicky and Miller presented a way to overcome their inherent drawbacks [HM03, HM04, WBMM06]. They introduced a family of pseudo-random algorithms for decentralized data distribution that map replicated objects to a scalable collection of storage servers or disks. These algorithms are primarily designed for very large-scale storage systems in which new disks are always added in terms of weighted homogeneous clusters. Moreover, all of an object's replicas are stored strictly disjoint (for example, some peer-to-peer systems, such as *OceanStore* [REG+03], do not make such guarantees). However, since all algorithms are laid-out to work for large-scale systems, a per-cluster weighting is performed rather than weighting individual disks which leads to a fair distribution with respect to clusters only. Furthermore, in case of system scaling, the redistribution of data objects solely considers uniform servers in weighted clusters, thus, fairness can only be achieved for uniform disk capacities making the scheme less usable for heterogeneous disks (but which is tolerable since they assume server addition in clusters only).

Another approach to obtain a scalable and fault-tolerant strategy that is furthermore able to handle non-uniform capacities was presented by Brinkmann et al. [BEHV05, BE07].

They show the coupling of a scalable data placement scheme called *SHARE* (c.f. Section 3.5.1 for details on SHARE) with a separate RAID 1 layer on top. This approach is implemented inside the storage virtualization solution *V:DRIVE* using SHARE as the core allocation algorithm to become scalable. In particular, for a given replication parameter r, the V:DRIVE layer creates r virtual volumes from the physical storage devices which are then exported to the RAID layer for generating mirrored RAID volumes on top. However, although efficient, the inherent drawback of this generated stack of strategies is an increase in error-proneness because of a higher number of different but interacting strategies.

In addition to storing multiple copies for each data item, the use of parity information to tolerate a small number of failures has reached high attraction. The key idea is to introduce redundant information into each data stripe by keeping the bit-wise exclusive-or (XOR) information of the data blocks in the stripe in an extra parity block. Then, all blocks are stored on separated disks, and in case of a single disk failure, the XOR operation guarantees complete information recovery from the residual disks. Due to their simplicity, efficiency and low capacity consumption, parity based schemes have widely been applied, like in RAID level 4/5 [PGK88, CLG+94] or numerous video servers [BGMJ94, SM98a, TPBG93, VRG95].

As simple parity approaches only prevent from one erasure, advanced schemes, like the *EVENODD* strategy [BBBM94], are able to tolerate two missing disks. This is accomplished by keeping a matrix in which the regular RAID level 5 layout is encoded and parity information is added for each diagonal of the matrix. Moreover, an EVENODD approach can be extended to tolerate up to t arbitrary simultaneous disk failures, but in this case, $t \cdot \ell^{1-1/t}$ disks are needed for keeping parity information [HGK+94], where $\ell < n$ denotes the stripe length. Thus, with respect to a fast reconstruction, this approach is only of theoretical interest. As a consequence, more complex erasure codes even for storage networks, like SANs or peer-to-peer networks, have become important that are able to recover from a higher number of disk failures (but which we leave out for the moment and refer to Chapter 4 for more details).

However, all known erasure coding schemes mainly focus on an efficient and almost optimal data recovery in case of erasures. But as we see in Chapter 4, this induces a negative update behavior in case of information modifications. Therefore, to overcome this drawback, we introduce so-called *Read-Write Codes* that feature an improved update flexibility for stored codewords and which are very useful for SANs.

Heterogeneity

Obviously, an easy solution to face heterogeneity could be to divide disks into clusters of equal characteristics, as proposed by Honicky and Miller [HM03, HM04, WBMM06].

Since inside each cluster, the disks appear to be uniform, the aforementioned strategies can be applied without explicitly regarding heterogeneity. However, this principle comprises some drawback. If newer and perhaps faster disks are added, a significant improve in access performance is only given for the subset the new disks were added to. Thus, response time strongly correlates to the clusters and the clustering policy, respectively. As an advantage, if access patterns are known in advance and remain stable over time, such clustering could be used to dissolve access hot spots. Unfortunately, such assumption is not very realistic.

Another principle that bases on grouping is given by the *HERA* approach, a heterogeneous extension of RAID [ZG00]. In HERA, a strategy called *Disk Merging* is introduced that was first proposed in [ZG97, Zim98] and which constructs a logical collection of uniform sized volumes from an array of heterogeneous physical disks. Each logical volume is created by aggregating a certain number of fixed sized fractions of the bandwidth or capacity provided by different physical disks. For example, in Figure 2.3, the logical volume number 2 (d_2^ℓ) is fed by fractions from the disks 0, 1, and 2 (d_0^p, d_1^p, d_2^p).

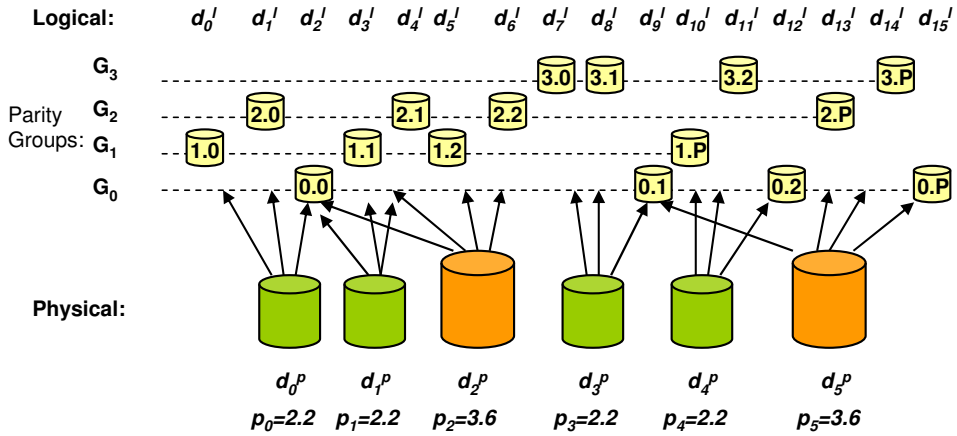

Figure 2.3: HERA: Heterogeneous Extension of RAID

According to a given parity scheme, the volumes are then grouped into different parity groups. As a main constraint, it must be considered that two logical volumes may be mapped to the same physical device, thus, they become dependent because if the physical disk fails, both volumes are affected. Consequently, since with simple XOR exactly one failure can be tolerated, two different volumes may not be mapped to the same parity group. To show the reliability of the system, the behavior is modeled by different processes, like the failure process and the reconstruction process. Because the MTTF of any disk is given by vendors, the time of these processes can be estimated; using a Markov model [Gib99], the mean time to service loss (MTTSL) can be derived for such heterogeneous environments. It was shown that the system reliability is only a factor of approximately 10 away from the best possible configuration, namely the clustering of

identical disks, as outlined above. However, the scheme itself and its provided performance strongly depends on the control of an administrator and is thus unusable for large environments.

In *AdaptRAID* [CL00, CL01], the regular RAID layout is extended such that heterogeneity in terms of different capacities can be handled. We only sketch the initial idea here which is as follows. The disks are sorted according to an increasing capacity. Then, the goal is to put as many stripes of length $\ell = n$ on all disks as possible. As soon as some k disks are saturated by data blocks, stripes of length $n-k$ are mapped onto the remaining $n-k$ disks, and so forth. The main drawback of this approach is that, depending on different stripe lengths, access to data stored in the upper part of a disk is faster than in the lower part. To overcome this problem, a second layer of different data patterns must be defined carefully which compensates the gaps between the initial patterns. Access to the data blocks that are encapsulated in fix sized stripes then is twofold making it hardly usable if the environment is allowed to scale.

At last, we briefly sketch the *RIO (Random I/O) Mediaserver* [SM98a, SM98b], a generic multimedia storage system capable of efficient, concurrent retrieval of many types of media objects. The RIO server defines a randomized distribution strategy and supports real-time data delivery with statistic delay guarantees. Data is placed to randomly chosen positions on randomly chosen heterogeneous disks. Disk bandwidth and capacity are coupled by a bandwidth-to-space ratio *BSR*, and disks with a high BSR can deliver data faster than some with lower BSR. To better utilize the preferred disks, replication is used such that copies of data blocks are assigned to these disks. The only remaining problem then is to identify the degree of redundancy to introduce into the system to sustain a given load but which can be estimated by simple calculations (that we leave out at this point).

Importantly, this scheme already emphasizes that randomization performs best when having to cope with heterogeneous disk such that disk utilization and overall system performance can optimally be provided.

For the rest of this work, note that additional related work that bears a higher relation to the results representing the core of this thesis can be found located closely to the particular strategies.

CHAPTER 3

Random Allocation of Data Copies

In this first scenario, we refer to an open storage management scheme as described in Section 1.1.2 in which we consider a SAN as a single, flat storage solution consisting of storage devices of arbitrary capacities. Furthermore then, we investigate data placement approaches that use *replication* for obtaining fault-tolerance in storage networks. As we already mentioned, replication is the easiest way for introducing redundancy into a storage system because the only thing to do is to store r identical copies of each data item separately on r different disks what makes the system capable of tolerating up to $r-1$ erasures. Surely, the storage of $r-1$ additional replicas of some data block implies a storage overhead of a factor of $r-1$, but this waste of capacity is outweighed by the inherent simplicity of the scheme because, basically, the only challenge to cope with results in controlling the separate storage of the r copies on different disk. Depending on this inherent simplicity, replication has become very popular in many application areas because it neither depends on complex control structures nor on any operational overhead, as, for instance, given with erasure coding (see Chapter (4)). Again, the most prominent representative using replication in the context of storage networks is the well-known RAID scheme whose level 1 definition (also known as *mirroring*) applies replication in a pure deterministic manner, and practical usage has shown that, as long as the observed environments do not exceed some certain size, the RAID 1 scheme is quite efficient.

However, if systems grow, the attempt to control any deterministic allocation regarding the redundancy and fairness conditions becomes pretty challenging. Moreover, if, besides the required redundancy, additional parameters like arbitrary disk capacities and/or dynamic system behavior come into play, with a deterministic scheduling, the costs, respectively computational complexity for the allocation considering all demands listed in Section (1.2), rise significantly with the size of the storage environment, what is dismal. Thus, alternatively, a well-known and very efficient paradigm to overcome these drawbacks is given by *randomization* which has become a standard approach in algorithm design since efficiency and simplicity are the main features of randomized algorithms.

The usage of randomization is not that new, and probabilistic based schemes have been applied successfully to conveniently model load balancing in many allocation problems, such as job scheduling, hashing, network routing or data allocation in e.g. distributed hash tables (DHT) [Sch01, DKM$^+$88, KLM92, ACMR95]. Again, randomized placement schemes are often much simpler than their deterministic counterparts, efficient to use, redundantize the need for complex and sensitive centralized control (as required by deterministic schemes) and provide efficient results even for non-uniform settings.

Surely, one of the main differences between deterministic and randomized strategies is that, in general, going from a deterministic scheduling that becomes very costly for large systems to short computations of randomized algorithms is often paid for by the risk of computing a wrong, respectively erroneous outcome, what is also undesired. Fortunately, for all our strategies, we can show that, besides being redundant, they appropriately satisfy the desired fairness condition, and the deviation of the disk loads resulting from the randomized allocation compared to an optimal data layout is considerably small, with high probability. Note that, when using a randomized placement, the higher the concentration of the load of some disk around the expected value (which equals the load assigned to the disks by an optimal algorithm) the more evenly the data layout becomes. Again, this basic feature prepares the ground for efficient parallel access to the data stored on slow mechanical based devices because by storing the data in a randomized fashion among all accessible disks, all I/O requests are also fragmented and evenly distributed over the storage network.

Furthermore, unlike any deterministic approach, randomization offers high potential to *scalability* (see e.g. [JK77, KLMadH96, BMadHS97, Sch01]) which has become an important property of modern SANs over the last decade to efficiently face the increasing system sizes and huge amounts of stored data making any downtime or interruption of service intolerable, and therefore crucial for a company. Hence, to keep the system at high operation, the costs for restoring an efficient data layout after any change in the set of disks should be minimized. To guarantee this, a data placement scheme should be *adaptive*, that is, in case of changes, it should guarantee to redistribute only a minimum amount of data while after redistribution, a fair data layout should again be achieved. This task is already of great challenge in a non-fault-tolerant and homogeneous setting considering uniform capacities only, but in a heterogeneous environment, this challenge further increases, and finally, if redundancy must furthermore be taken into account, this task turns to be pretty hard. Nevertheless, depending on the inherent properties of randomization, this demand can be tackled more conveniently by a randomized strategy. A good introduction to the design and analysis of a broad bouquet of randomized algorithms is given in [MR95, HZ05]. Furthermore, Scheideler [Sch00] presents several probability methods that can be used efficiently for different types of problems, like e.g. routing in networks, job scheduling, coloring hypergraphs and data distribution processes.

Outline of this Chapter

For what follows, we investigate and analyze novel randomized replication strategies under fault-tolerance consideration, and the major problem on what we are interested in this chapter is to always find a fair load balancing for storage networks that may consist of arbitrary disk capacities. Obviously, this is a pretty challenging task because, compared to usual randomized placement algorithms which allocate all data items independently from each other, regarding the redundancy condition at any point in time induces dependencies on the allocation of any two identical copies, thus, making the allocations no longer independent from each other. For dynamic systems that are often allowed to undergo changes in the number of connected devices caused by the addition or removal of accessible disks due to maintenance, capacity extensions or disk failures, this problem becomes even harder.

Since companies run static as well as dynamic systems, we divide our observations into two parts. At first, we consider static storage environments only, while the second part also tolerates dynamic system behavior. This implies that, in the first part, we can restrict our analysis on the data layout, i.e. on the load of the disks after allocation, whereas in the second part, we also have to consider the replacement of data blocks in case of disk additions or removals.

In more detail, for the static scenario, we analyze strategies that are closely related to those applied for the *classical resource allocation problem*. In that problem, a system is considered in which for each arriving data block a scheduling algorithm chooses the location of the block on-line and uniformly at random from a fixed number of identical resources (e.g. disks of same size). Since in this scenario, both the resources and the data blocks are supposed to be uniform, and additionally, the blocks are considered to be pairwise independent (i.e. no redundancy is assumed), near-optimal allocations can be obtained by using algorithms which base on the widely known *Balls-into-Bins* model, a simple model that has become very prominent in the load balancing community (see e.g. [JK77, RS98, ABKU00, BCSV00, Sch00]). In short words, in a balls-into-bins model, one sequentially allocates a set of m independent balls (representing tasks, jobs, data blocks,...) at random to a set of n bins (servers, processors, disks, ...) such that the maximum number of balls in any bin is minimized. In the recent years, many useful and efficient results have been obtained by this simple scheme, most of which hold *with high probability*[1]. However, unfortunately, nearly all of the known results only consider uniform capacity distributions, and more important, none of the results covers the redundancy condition we address in this work (we return to the balls-into-bins approach in more detail in Section 3.4).

[1] We denote an event A to occur with high probability (w.h.p.) if $\Pr[A] \geq 1 - n^{-\ell}$ for an arbitrarily chosen constant $\ell \geq 1$.

The dynamic scenario is significantly harder as to maintain fairness, any given change in the capacities of the disks caused by disk additions or removals implies some redistributions on the stored data blocks. Thus, to avoid any intermediate state in which at least one of the given requirements is violated, the applied placement scheme is forced to re-establish the fair and redundant data layout after any step in the redistribution phase.

3.1 Preliminary Section

Summarizing the descriptions in the previous section, we are interested in finding efficient randomized algorithms for some sort of *redundant resource allocation problem* for arbitrary capacities under either static or dynamic conditions. This section is dedicated to give useful abstractions for addressing this problem in a theoretical manner. First, we introduce a basic model of the considered storage networks and then define essential attributes on which we concentrate in the subsequent sections to evaluate the resulting load balancing after the random data allocation. That is, as we are mainly interested in the layout quality of the allocations throughout the whole chapter, we solely focus on the *insertion* of data blocks and omit the *deletion* of already stored blocks. In addition, we only consider a finite number of data blocks to be allocated, thus, we restrict ourselves to a *finite* allocation problem what is different to an *infinite* allocation model which is more suitable for problems such as job allocation on processors or video-on-demand scenarios. (Moreover, for the rest of this chapter, we use the terms *bins, disks, nodes* interchangeably, and this also holds for *balls, data blocks* and *data items*.)

3.1.1 The Basic Model

First of all, for the following descriptions, we define the set $[c] := \{1, \ldots, c\}$ for any $c \in \mathbb{N}$. Then, for all observed storage systems, we consider arbitrary but fixed numbers $M, N \in \mathbb{N}$ sufficiently large such that the set $V = [N]$ denotes the address space of all possible disks (bins, nodes), and $U = [M]$ is the address space of all possible balls (data items) in the system. Given these sets, we suppose that, at any time, $m \leq M$ is the number of balls currently in the system, and $n \leq N$ always denotes the current number of accessible disks. Naturally, as storage networks consist of a collection of hard disks each covering of a huge number of data blocks, we assume $m \gg n$, and since our distribution schemes pay no attention to the content of any given data block, this also implies to consider the balls as *uniform*, that is, their weights are normalized, i.e. each ball $j \in U$ has weight $w_j = 1$. Furthermore, due to redundancy requirements, let each data item be represented in the system by $r \in [n]$ identical copies, with r called the *replication parameter* (for a non-redundant system, we set $r = 1$). Since we assume r to be fixed for all data items and we always consider $m \leq M$ balls currently in the system, let $[k]$ be the set of original, pairwise

distinct data items, for $k = \frac{m}{r} \in \mathbb{N}$. Generally, any given I/O operation on the data blocks (i.e. read or write access) is referred to as *data access* or *data request*.

For a formal description of our storage network model, we refer to some definitions from the thesis of Kay Salzwedel [Sal04].

Definition 3.1. (Storage Network) *A **storage network** $S(n, C[n])$ is a collection of $n \leq N$ accessible disks $[n] \subseteq V$. Each disk i is characterized by a capacity $C_i \in \mathbb{N}$ describing the total number of (uniform) balls the disk can store. Let $C[n] = (C_1, C_2, \ldots, C_n)$, and the total capacity of $S(n, C[n])$ is $C_{total} = \sum_{1 \leq i \leq n} C_i$.*

*A storage network $S(n, C[n])$ is called **homogeneous (or uniform)** if all disk have the same capacity, that is, for any two disks $1 \leq i, j \leq n$, it holds that $C_i = C_j$. Otherwise, the system is called **heterogeneous (or non-uniform)**.*

In Figure 3.1, we depict an arbitrary storage network consisting of a number n of disks connected by some dedicated network. As further given in the picture, the disks may feature different characteristics, like e.g. the capacity C_i or the bandwidth B_i stating the average transfer time when accessing the disk by an I/O operation. For the purpose of illustration, we will restrict ourselves to disk capacities only throughout this thesis because disk utilization is more self-evident than throughput utilization.

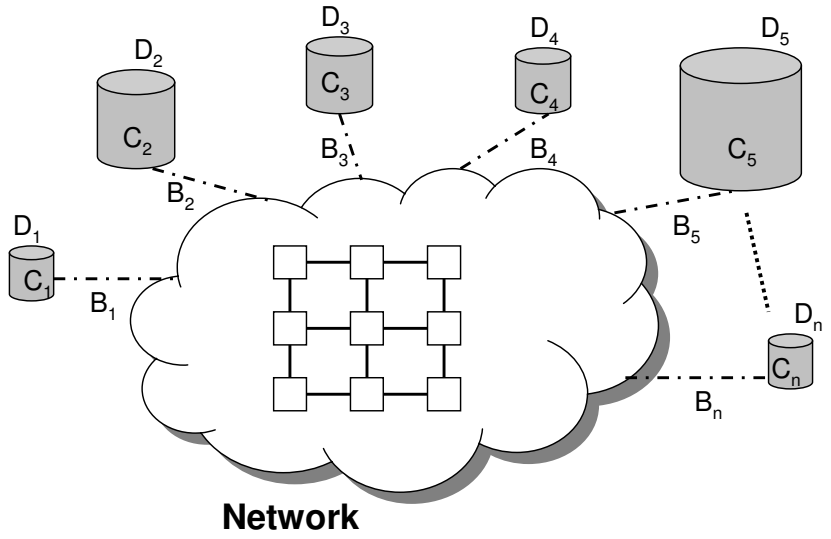

Figure 3.1: A storage network with non-uniform characteristics (capacity, bandwidth).

Since throughout this chapter, we consider random allocations on the disks of a storage network, $S(n, C[n])$ is considered as the given sample space for allocation on which we can model an appropriate probability distribution by the following *shares* of the disks.

Definition 3.2. (Share) *Given a storage network $S(n,C[n])$, the **share** of a disk i is defined by its relative capacity*

$$c_i = \frac{C_i}{C_{total}} \in [0,1].$$

Thus, $\sum_{1\leq i\leq n} c_i = 1$. In other words, the share of each disk currently in the system is a normalized measure of the heterogeneity in a storage network. Moreover, given the share tuple $\bar{c} = (c_1, c_2, \ldots, c_n)$ for the disks in $S(n,C[n])$, obviously, $(S(n,C[n]),\bar{c})$ defines an appropriate probability space for random allocation in which the sampling of the disks is supposed to be done independently at random (at a first glance, the probability space $(S(n,C[n]),\bar{c})$ defines a base for our investigations).

Given the share tuple $\bar{c} = (c_1, c_2, \ldots, c_n)$, we can now define our random allocation problem for a storage network $S(n,C[n])$.

Definition 3.3. (Redundant Allocation Problem) *Let $S(n,C[n])$ be a storage network with a set $[n]$ of accessible disks and capacities given by the share tuple $\bar{c} = (c_1, c_2, \ldots, c_n)$. Furthermore, consider a set $[k] \subseteq U$ of data items and a replication parameter $r \in [n]$ that holds for every $d \in [k]$.*

*Depending on r, let $(f_1, f_2, \ldots, f_r)$ be an **r-tuple** of allocation functions $f_i : [k] \to [n]$, $i \in [r]$, satisfying the following condition:*

$$\forall i, j \in [r], i \neq j, \forall d \in [k] : f_i(d) \neq f_j(d).$$

*As for each $d \in [k]$, $(f_1, f_2, \ldots, f_r)$ generates an **allocation tuple** $(f_1(d), f_2(d), \ldots, f_r(d))$, we consider the set $M_k^r = \{(f_1(d), f_2(d), \ldots, f_r(d)) \,|\, d \in [k]\} \subseteq [n]^r$ of allocation tuples. Then, let $F = (f_1, f_2, \ldots, f_r)$ be a **redundant allocation function** defined as*

$$F : [k] \to M_k^r,\ F(d) = (f_1(d), f_2(d), \ldots, f_r(d))\ \ \forall d \in [k].$$

*Since $|M_k^r| = k$, we obtain a total of $m = k \cdot r$ different allocations (balls) in the system. Then, $S(n,C[n])$, the share tuple $\bar{c}$, and the random allocation function F define a **redundant allocation problem** on the probability space $(S(n,C[n]),\bar{c})$. The allocation problem is **finite** as M_k^r is finite.*

Again, since we always assume $m \leq M$ balls in the system, we fix $m = k \cdot r$ in the following. Additionally, as considering the balls to be uniform, we introduce the ***load*** ℓ_i of a given disk i as the number of data blocks allocated to it.

Now, given the definition of the redundant allocation problem, we can formulate a suitable definition for a redundant data distribution strategy as we require in the following descriptions.

Definition 3.4. (Redundant Data Distribution Strategy) *A **redundant data distribution strategy** for the redundant allocation problem defines a disjoint mapping $F : [k] \to M_k^r$ for any number $m \leq C_{total}$ of blocks such that for each disk i the load $\ell_i \leq C_i$.*

To address the fairness condition on the disks in $S(n,C[n])$ in a formal manner, we apply a set of binomially distributed random variables $X_1,X_2,\ldots,X_n$ such that for each disk $i \in [n]$, let X_i denote the load ℓ_i of disk i. Then, for a total of m data blocks in the system and any given integer $k \in \mathbb{N}$, it holds: $\Pr[X_i = k] = b_{m,c_i}(k) := \binom{m}{k} c_i^k (1-c_i)^{m-k}$ and $\mathrm{E}[X_i] = m \cdot c_i$ is the expected value.

With this random variables, we are now able to measure the fairness property of the defined redundant data placement scheme (sometimes also referred to as *space balance*) (c.f. Section 1.2).

Definition 3.5. (Fairness) *For a redundant allocation problem, let $X_1,X_2,\ldots,X_n$ be binomially distributed random variables such that X_i denotes the load of disk $i \in [n]$. Then, a redundant data distribution strategy is **fair** if for all disks $i \in [n]$*

$$\ell_i = \mathrm{E}[X_i] = r \cdot k \cdot c_i.$$

*Moreover, since we require that we come close to optimal, the fairness condition above is called **tight** if the expected load of disk i is in range*

$$\ell_i = (1 \pm \varepsilon) \cdot r \cdot k \cdot c_i$$

w.h.p., where $\varepsilon > 0$ can be made arbitrarily small.

Recall that the fairness property ensures the utilization of the storage network, and as each disk's charging level corresponds to its share, a tight fair data distribution enables the system to get filled up almost completely.

Up to this point, the given definitions enable us to characterize a redundant allocation only for a static storage network, that is, the configuration of the storage network does not change over time; the only dynamic aspect that comes into play with respect to such systems is given if the number m of blocks in the system changes.

However, this is not yet sufficient as we are further interested in an appropriate formulation for a dynamic setting in which a given configuration of current disks may change due to altering space requirements. Thus, in the following, we formalize the adaptivity condition for a redundant allocation strategy, that is, we enable a storage network to increase or decrease in the number of disks. The major algorithmic challenge coming along with this additional property is to ensure both the fairness and redundancy condition at any time. A natural measure for the efficiency of a distribution strategy in terms of adaptivity is the number of blocks that have to be redistributed to ensure the fairness property in case of changes in the configuration. This measure determines how long the storage system works in a degraded mode and in which performance losings must be considered. In the following, we call operations that induce a replacement of blocks *change operations*.

As usual, we apply *competitive analysis* (see Section 3.1.3.5 for details) which is a well-known method to model and evaluate this dynamic process that we can also comprehend as an online algorithm. In short words, competitive analysis compares an online algorithm with an optimal offline strategy. In particular, for our considerations, we would require an optimal algorithm to ensure fairness after each change operation ω in the system implying that, for instance, if ω is triggered by the addition of a disk, any algorithm has to replace at least $c_{n+1} \cdot m$ blocks. We call an adaptive distribution strategy *c-competitive* if it requires at most c times the number of replacements performed by the optimal offline algorithm for any sequence of change operations. Now, we can formulate a useful definition for an adaptive redundant data placement strategy as we consider in the following sections.

Definition 3.6. (Adaptive Redundant Data Distribution Strategy) *Consider a storage network $S(n,C[n])$ and any change operation ω. We call a redundant data distribution strategy **adaptive** for $S(n,C[n])$ if it is able to restore the fairness and redundancy property on $S(n,C[n])$ after any change in the number of current disks n induced by ω.*

*It is **c-competitive** if the number of blocks necessary to restore fairness is at most c times the number of blocks an optimal strategy has to replace, on expectation.*

Obviously, to appropriately model and analyze the dynamic case observed in Section 3.5, we suppose an extended share tuple $\bar{c} = (c_1, c_2, \ldots, c_N)$ for $N \geq n$ and a given storage network $S(n,C[n])$. Furthermore then, all disks i not currently in the network are assigned are share $c_i = 0$. Hence, we consider an increase in the number of disks up to an upper bound N. Then, given two share tuples $\bar{c} = (c_1, c_2, \ldots, c_N)$ and $\bar{c}' = (c'_1, c'_2, \ldots, c'_N)$, for any data distribution strategy to be adaptive, it must be able to handle any capacity change from $\bar{c}$ to $\bar{c}'$. Certainly, it is easy to see that every storage strategy that wants to preserve fairness has to replace at least a

$$\sum_{i:c_i > c'_i} (c_i - c'_i) = \frac{1}{2} \sum_i |c_i - c'_i|$$

fraction of the data in the system.

Regarding all given definitions, we can, roughly speaking, formulate the *redundant allocation problem*, on that we concentrate throughout this chapter as follows:
Is it always possible to find an efficient, fair, redundant (and perhaps) adaptive random data distribution strategy for arbitrary capacities such that, at any time, best performance provided by parallel access as well as fewest resource consumption can be guaranteed ?

Before, in the rest of this chapter, we concentrate on the description and analysis of our novel data distribution strategies that address the given problem appropriately, either for a static or dynamic setting, we first give some preliminaries in the next section which we require for the modeling and the analysis of those allocation schemes.

3.1.2 On the Practical Realization of the Allocation Function F

Obviously, finding a positive answer to the formulation of the redundant allocation problem can be directly translated into finding an appropriate redundant data distribution strategy $F = (f_1, f_2, \ldots, f_r)$ for either a static or dynamic system. Thus, for the rest of this chapter, we aim at finding a suitable translation of the theoretic definition given in the previous section into some practical realizations of F for any given storage system that can easily be implemented inside a respective application. In particular, this means that such realization always has to ensure that

- any two functions f_i, f_j, $i \neq j$, in the r-tuple $(f_1, f_2, \ldots, f_r)$ allocate a given data block $d \in [k]$ on distinct bins $\ell, o \in [n]$, $\ell \neq o$,
- each of the functions f_i in $(f_1, f_2, \ldots, f_r)$ is efficiently computable, and
- for a given number $m \leq C_{total}$ of blocks, $(f_1, f_2, \ldots, f_r)$ yields a fair data layout up to some factor $(1 \pm \varepsilon)$ for each disk.

Clearly, by allocating data blocks in a given storage network $S(n, C[n])$, the function F also implements the *dictionary functions search*, *insert* and *delete* for the data blocks on $S(n, C[n])$ (again, we solely concentrate our analysis on the data layout only, that is, we do never consider the deletion of blocks). With respect to a technical implementation of F, at a first glance, it seems to be a good idea to realize the functions f_i in F by r different hash functions $h_1, h_2, \ldots, h_r$ because hash functions can be efficiently computed, and more importantly, by using hash functions, the expected complexity of the dictionary functions can be reduced to $O(1)$ (in Section 3.1.2.1, we go into further details on hash functions).

However, the proposed idea imposes some severe problems. First of all, the usage of r independent hash functions cannot adequately ensure a perfectly disjoint allocation of all identical copies, but what we strictly require, because all of them hash each data block into the same range $[n]$ of disks. Second, depending on r, the number of applied hash functions may be too small to achieve fairness as defined in the previous section. At last, all of those hash functions must be kept in memory, thus, if r is supposed to be large, this implies a waste of memory, what is dismal. Hence, it is easy to see that, since all our introduced strategies are randomized algorithms, this makes a separate allocation of identical copies become a pretty hard challenge. Moreover importantly, the redundancy property must always be ensured even in dynamic storage environments.

Therefore, as we show by the descriptions in this chapter, there is considerably more work to do to realize F for a given storage environment. Recall that each function F has to be chosen such that the elements from the set M_k^r are allocated to $S(n, C[n])$ in a way that, on expectation, fairness can almost perfectly be achieved. Nevertheless, it is a good idea

to consider hashing for allocation, and as we show in the following description, there are useful *universal hash functions*, called *k-universal hash functions* for some parameter k, that ensure a suitable distribution of the data blocks on the set of disks and which underly all of the strategies introduced in this chapter.

We start our descriptions with introducing a basic design paradigm for randomized online algorithms that is applied by the functions and strategies presented in the following. After that, we briefly sketch universal hashing (because it belongs to folklore) before we introduce k-universal hash functions. At last, we prove that for both static and dynamic storage environments, $O(\log n)$-universal hash functions are a good choice to obtain a useful dispersal of blocks w.h.p.

Design Paradigm: Foiling the Adversary. To find a suitable realization for all random allocation functions F in this chapter, we apply the well-known design paradigm for randomized algorithms, the *method of foiling the adversary*, also called the *method of avoiding the worst-case problem instance* (cf e.g. [HZ05]). The paradigm considers the design of an algorithm as a game between two players: a designer who tries to design an efficient algorithm for a given problem, and an adversary who for any designed algorithm tries to construct an input on which the algorithm fares poorly, that is, the output is not efficient or incorrect. Typically, in the analysis of deterministic algorithms, the adversary argument establishes a lower bound on the worst-case complexity, and, in general, only those algorithms that run efficiently on *every* given input are considered to be efficient. As the designer has to present his algorithm at first in the game, the adversary has some advantage since, knowing the algorithm, he can find a hard problem instance for the algorithm. This situation can essentially change if the designer designs a randomized algorithm A, that is, the adversary does not know which of the possible runs of the algorithm A will be chosen at random. Therefore, the paradigm is very suitable for problems for which

1. every deterministic algorithm (strategy) has some worst-case instances at which the algorithm is not able to efficiently compute the correct (or optimal) output,

2. but there exists a class of deterministic algorithms such that, for every problem instance, most algorithms of this class efficiently compute the correct result.

If (2.) holds, then for any given input instance, one can pick an algorithm from the class at random and expect to get the correct result efficiently with reasonable probability. As we will see, the following universal hashing approach describes a successful application of this paradigm.

3.1.2.1 Excursus: Universal Hashing

Hashing is a well-known method of dynamic data management. In particular, given a data structure $T = [N]$, called *hash table*, which consists of $N \in \mathbb{N}$ addressable slots to which direct access is provided[2] as well as some data item addressed by its key k, hashing efficiently performs the dictionary functions *search(T,k)*, *insert(T,k)* and *delete(T,k)*. The key k is taken from a universe U of all possible keys that may be a finite subset of $\mathbb{N}$, or $U = \mathbb{N}$. The major goal of hashing then is to perform the dictionary operations in $O(1)$ time, what is different to other dynamic data structures such as AVL-trees or B-trees performing these operations in logarithmic time in the worst case.

We furthermore consider some certain application by which we are usually given a finite input set $S \subseteq U$ of size $M \approx N$, and the aim is to save S in T such that the directory operations can be provided as efficient as possible on T. It is easy to see that, to achieve the desired efficiency for the dictionary operations, the task now is to determine a mapping $h : U \to T$, called a *hash function*, on which we pose the following requirements:

- h can be efficiently computed,
- h maps every input set $S \subseteq U$ in a 'well-dispersed' fashion to T in a way that, for all $i \in [N]$, the cardinality of the set $T(i) = \{d \in S \,|\, h(d) = i\}$ is in $O\left(\frac{|S|}{|T|}\right)$.

Unfortunately, the main problem arising with respect to the mapping h is that **we cannot influence the choice of** S as well as we do not have any preliminary information about S; the set S is given completely by the application (user), thus, we are not able to predict anything about it. Moreover, one can show that for every hash function h, there exist "bad" sets S, all of whose elements are mapped by h to some slot i of T (for instance, consider S to be a subset of the set $U_{h,i} = \{d \in U \,|\, h(d) = i\}$ (see [HZ05])). On the other hand, for each given set S, there exist useful hash functions that appropriately distribute the keys of S among the slots of T.

Summarizing this, we have an exemplary situation calling for applying the 'foiling the adversary' paradigm. For every choice of h, the adversary can construct arbitrarily bad inputs S, but on the other hand, there are many hash functions that assure the best possible distribution for most inputs S. Following the paradigm, the only possibility then to distribute all (and also bad) input sets evenly among the slots of T is to choose a hash function h uniformly at random from a suitable set H of hash functions. In other words, unlike assuming the input set S being chosen uniformly at random from the universe U, we rather suppose a class H of hash functions in which these functions are evenly distributed. Obviously, for most input sets, on average, this sampling guarantees an even distribution of the keys in T w.h.p. The considered 'suitable' classes H are widely-known as *universal classes of hash functions* which are defined as follows:

[2] One is able to examine any slot of T in time $O(1)$.

Definition 3.7. (Universal Hash Functions): *Let H be a finite set of hash functions from U to T ($|T| = N$). The set H is called **universal** if for each pair of elements $x, y \in U$, $x \neq y$,*

$$|\{h \in H \,|\, h(x) = h(y)\}| \leq \frac{|H|}{N}$$

holds, i.e. at most every N-th hash function from H maps x and y to the same slot of T.

That is, for any fixed pair of keys $x, y \in U$, $x \neq y$, it holds the following condition

$$\Pr_H[h(x) = h(y)] \leq \frac{1}{N}. \tag{3.1}$$

As a major result, it can be shown that for an arbitrary finite set $S \subseteq U$ of keys and a universal class H of hash functions from U to T, the expected number of keys in any slot of T is smaller than $1 + \frac{|S|}{|T|}$ [HZ05]. (We skip going into further detail at this point because universal hash function are subject to any basic course on algorithms and can furthermore be found in e.g [HZ05, CLRS01].)

As we can see, the concept of universal hash functions provides a useful mechanism to obtain an appropriate distribution of the data items among the storage devices. It only remains to show that such classes exist at all, and if so, to find a universal class H of hash functions that is suitable for our requirements. The first question can be answered positively by the following lemma.

Lemma 3.8. *Let U be a finite set. The class $A = \{g \,|\, g : U \to T\}$ of all functions from U to T is universal.*

We omit the proof here (but which can be found in e.g. [HZ05]). Now, we know that classes of universal hash functions exist but, unfortunately, many of them suffer from huge size. For instance, the number of all functions in A is[3] $N^{|U|}$ what, obviously, makes A impossible for real applications since

1. A is too large[4] to be able to perform a random choice of an h from A efficiently, and
2. most functions from A can be viewed as random mappings in the sense[5] that they do not have any essential shorter description than their full table representation. Such functions are not efficiently computable.

Instead, we would like to have a universal class H of hash functions such that

(i) H is small (at most polynomial in $|U|$), and

[3] One has to assign one of the N values from T to each element of U.

[4] Observe that U is already very large, thus, $N^{|U|}$ is usually substantially larger than the number of protons in the known universe.

[5] c.f. [Hro03].

(ii) every hash function from H is efficiently computable.

The most prominent universal hash functions satisfying the formulated wishes are the linear hash functions $h_{a,b} : U \to T$ defined by

$$h_{a,b}(x) = ((ax+b) \bmod p) \bmod |T|,\ a,b \in U,\ p \text{ prime.}$$

for every $x \in U$. The corresponding universal class of hash functions is

$$H^p_{\text{lin}} = \{h_{a,b} \,|\, a \in [p-1] \text{ and } b \in \{0,1,\ldots,p-1\}\}$$

of size $|H^p_{\text{lin}}| = p \cdot (p-1)$. Unfortunately, as we can see from the definition of $h_{a,b}$, linear hash functions guarantee the desired properties only for any two elements $x, y \in U$ (see [HZ05]). Instead, for our further analysis in this thesis, we require what we call *k-independence*, that is, for any given set of k pairwise distinct keys from U, the hash function guarantees that Condition 3.1 holds also (independently) for all k elements. This demand is satisfied by the following *k-universal hash functions* (the following is taken from [MadH06]; further references are [CW77, DKM$^+$88]) (recall $[M]$ and $[N]$ to be the address spaces of the balls and bins, respectively).

Definition 3.9. **(k-Universal Hash Functions):** *Consider $H \subseteq A = \{g \,|\, g : [M] \to [N]\}$, $k \in \mathbb{N}$. H is a **k-universal** class of hash functions if, for all $\ell \leq k$, all pairwise distinct $x_1,\ldots,x_\ell \in [M]$ and all $j_1,\ldots,j_\ell \in [N]$, it holds that*

$$\Pr[h(x_i) = j_i,\ \forall i \in [\ell]] \leq \left(\frac{2}{N}\right)^\ell$$

To simplify our further analysis, we weakened the given bound for the probability of the random function h that, if considered to be an arbitrary random function from the set A, would be $\left(\frac{1}{N}\right)^\ell$. However, for our assumptions, $h \in H$ is sufficiently random, and if we merely consider $\ell \leq k$ input values, h nearly behaves like a real random function.

In what follows, we prove that some special classes of polynomials are universal hash functions. Let M be an arbitrary but fixed prime. Then, we can define the following:

Definition 3.10. *Suppose $d \in \mathbb{N}$. For a tuple $\bar{a} = (a_d,\ldots,a_0) \in [M]^{d+1}$ of coefficients, let*

$$h_{\bar{a}}(x) := \left(\left(\sum_{i=0}^{d} a_i \cdot x^i\right) \bmod M\right) \bmod N, \text{ for } x \in [M].$$

Furthermore, we define $H^d_N := \{h_{\bar{a}} : [M] \to [N] \,|\, \bar{a} \in [M]^{d+1}\}$.

That is, the class H^d_N consists of all polynomials of degree-bound d taken from the finite field $\mathbb{Z}_N$ (for every prime p, there exists a finite field $\mathbb{Z}_p$ with p elements (see e.g. [HP03, Rom96]) whose images have finally to be mapped into the set $[N]$.

Theorem 3.11. *The class H_N^d has the following properties:*

1. *each function $h_{\bar{a}} \in H_N^d$ can be evaluated in time $O(d)$, and*
2. *H_N^d is $(d+1)$-universal.*

Proof. Case (1) is obvious.
Case (2): For the tuple $\bar{a} \in [M]^{d+1}$ of coefficients, let $f_{\bar{a}} : [M] \to [M]$ be the polynomial

$$f_{\bar{a}}(x) = \left(\sum_{i=0}^{d} a_i \cdot x^i \right) \bmod M.$$

Since M is a prime, $f_{\bar{a}}$ is a polynomial of degree-bound d over the field $\mathbb{Z}_M$, and as we know, in a field, each polynomial of degree-bound d is uniquely determined by $d+1$ distinct points $x_0, \ldots, x_d$.

Now, consider an arbitrary but fixed ℓ, $1 \leq \ell \leq d+1$. Let $x_1, \ldots, x_\ell \in [M]$ be ℓ pairwise distinct input values for $f_{\bar{a}}$. Furthermore, let $j_1, \ldots, j_\ell \in [N]$ be possible output values 'modulo N', and let

$$A^\ell := \{h_{\bar{a}} \in H_N^d \,|\, h_{\bar{a}}(x_i) = j_i,\ \forall i \in [\ell]\}.$$

Then, it holds:

$$h_{\bar{a}} \in A^\ell \Longleftrightarrow h_{\bar{a}} = f_{\bar{a}} \bmod N, \text{ and } f_{\bar{a}}(x_i) \in \underbrace{\{j_i + \alpha \cdot N \,|\, \alpha \in [\left\lceil \frac{M}{N} \right\rceil]\}}_{=:B_i}, \forall i \in [\ell]$$

Since the $f_{\bar{a}}$ are polynomials of degree-bound d over a finite field $\mathbb{Z}_M$, each polynomial is uniquely determined by $d+1$ distinct values implying that the coefficients $\bar{a}$ of $f_{\bar{a}}$ are also uniquely determined. Furthermore, since we already fixed ℓ points of each polynomial in the beginning by choosing the ℓ point-values x_i and j_i, it remains to determine the residual $d-\ell+1$ distinct values to determine the polynomial uniquely. Thus, for each tuple $(k_1, \ldots, k_\ell) \in B_1 \times \ldots \times B_\ell$ with $f_{\bar{a}}(x_i) = k_i$, we can still determine further $[M]^{d-\ell+1}$ distinct values because the domain of $f_{\bar{a}}$ is $[M]$ and $d-\ell+1$ points are still left to choose.

From this, it follows that the number of hash functions that map to the chosen values j_i is

$$|A^\ell| = |B_0| \cdot \ldots \cdot |B_d| \cdot [M]^{d-\ell+1} \leq \left\lceil \frac{M}{N} \right\rceil^\ell \cdot [M]^{d-\ell+1}.$$

Moreover, since $|H_N^d| = M^{d+1}$ holds (because each tuple $\bar{a} \in [M]^{d+1}$ defines a different polynomial over the field $\mathbb{Z}_M$), if follows:

$$\Pr[h_{\bar{a}}(x_i) = j_i,\ \forall i \in [\ell]] = \frac{|A^\ell|}{M^{d+1}} \leq \frac{\left\lceil \frac{M}{N} \right\rceil^\ell \cdot [M]^{d-\ell+1}}{M^{d+1}} \leq \left(\frac{2}{N} \right)^\ell.$$

Thus, the class H_N^d is $(d+1)$-universal. ■

3.1.2.2 Analyzing the Distribution Quality of the Hash Functions

Given the above definition of k-universal hash functions, we can now analyze the distribution quality of the hash functions that we apply for our strategies. In particular, for a storage network $S(n,C[n])$ with arbitrary capacities, a set $P \subseteq [M]$ of data blocks (note that, for our analysis, we neglect redundancy, i.e. all data blocks are considered pairwise distinct), and a hash function h chosen uniformly at random from an $O(\log n)$-universal class of hash functions $H \subseteq A = \{g \,|\, g : [P] \to [n]\}$, we can state the following theorem.

Theorem 3.12. *Consider a storage network $S(n,C[n])$ with arbitrary capacities. Then, for every set P of $m \geq n$ data blocks, the hash function h provides an even distribution of the blocks among the set $[n]$ of bins, w.h.p*

Proof. Before we start, we introduce the following well-known inequality for binomial coefficients which we use for the analysis in Lemma 3.14.

Lemma 3.13. *For $k, \ell \in \mathbb{N}$, $0 \leq \ell \leq k$, it holds*

$$\binom{k}{\ell} \leq \left(\frac{k \cdot e}{\ell}\right)^{\ell}$$

Proof.

$$\binom{k}{\ell} = \frac{k \cdot (k-1) \cdots (k-\ell+1)}{\ell!} \leq \frac{k^{\ell}}{\ell!} = \frac{k^{\ell}}{\ell^{\ell}} \cdot \frac{\ell^{\ell}}{\ell!} \leq \frac{k^{\ell}}{\ell^{\ell}} \cdot e^{\ell}, \text{ since } e^{\ell} = \sum_{i=0}^{\infty} \frac{\ell^i}{i!} \geq \frac{\ell^{\ell}}{\ell!}.$$

■

Now, for every relative capacity value $c_i \in [0,1]$ in $S(n,C[n])$, fix some arbitrary bin $i \in [n]$ of size c_i and proceed as follows.

Assume the input set P to be of size $m = \alpha_i \cdot n \log n$ for some appropriate constant $\alpha_i \in \mathbb{N}_{\geq 1}$ such that P can be partitioned into equal sized batches $P = \{B_1, B_2, \ldots, B_{\alpha_i}\}$. To determine α_i, let $|B_k| = \frac{\tilde{c} \cdot \log n}{c_i}$ for all $k \in [\alpha_i]$ where $\tilde{c}$ is some constant which is chosen such that $\frac{\tilde{c}}{c_i} = n$ and that can easily be determined if we assume $c_i = \frac{1}{n}$ for a uniform network, and for non-uniform capacities, we consider $c_i = c' \cdot \frac{1}{n}$ for an appropriate constant $c' \geq 0$.

Now, consider each batch B_k to be allocated independently to the n bins by applying the hash function h. Then, we can state the following lemma concerning the distribution of the elements of B_k.

Lemma 3.14. *Let $S \subseteq B_k$. Then, for the load ℓ_i of some bin $i \in [n]$, the following holds*

1. $\Pr[\ell_i \geq c \cdot \log n] \leq \left(\frac{2e}{c}\right)^{c \cdot \log n}$ *for an appropriate constant c*

2. *For all bins $i \in [n]$ of size c_i, it holds:* $\Pr[\max_i\{\ell_i\} \geq c \cdot \log n] \leq \left(\frac{1}{n}\right)^{\ell}$ *for some arbitrary constant $\ell \in \mathbb{N}_{\geq 1}$ and an appropriate constant c*

3. *For all bins $i \in [n]$ of size c_i, it holds:* $\mathrm{E}[\max_i\{\ell_i\}] \in O(\log n)$

Proof. Case (1)

$$\begin{aligned}
\Pr[\ell_i \geq c \cdot \log n] &= \Pr[\exists A \subseteq S, |A| = c \cdot \log n, \forall d \in A : h(d) = i] \\
&= \Pr\left[\bigcup_{\substack{A \subseteq S \\ |A| = c \cdot \log n}} h(d) = i \ \ \forall d \in A\right] \\
&\overset{(*)}{\leq} \sum_{\substack{A \subseteq S \\ |A| = c \cdot \log n}} \Pr[h(d) = i \ \ \forall d \in A] \\
&\overset{(**)}{\leq} \binom{n \log n}{c \cdot \log n} \cdot \left(\frac{2}{n}\right)^{c \cdot \log n} \\
&\overset{(***)}{\leq} \left(\frac{n \log n \cdot e}{c \cdot \log n}\right) \cdot \left(\frac{2}{n}\right)^{c \cdot \log n} = \left(\frac{2e}{c}\right)^{c \cdot \log n}.
\end{aligned}$$

The estimate $(*)$ holds by Boole's inequality (see Section 3.1.3.1), $(**)$ holds since H is an $O(\log n)$-universal class of hash functions, and $(***)$ holds according to Lemma 3.13.

Case (2)

Now, we show that for all bins $i \in [n]$ of size c_i, the result from Case (1) holds w.h.p. That is, instead of focusing on a single bin only, we now consider all bins of size c_i and show that there is no bin having a higher load w.h.p. We denote by $\max_i\{\ell_i\}$ the maximum load of the bins we concentrate on. Furthermore, let A_i denote the event that the load of any bin i is $\ell_i \geq c \cdot \log n$. Then,

$$\begin{aligned}
\Pr[\max_i\{\ell_i\} \geq c \cdot \log n] &= \Pr[\exists i \in [n] : \ \ell_i \geq c \cdot \log n] \\
&= \Pr[A_1 \cup A_2 \cup \ldots \cup A_n] \\
&\overset{(*)}{\leq} \sum_{i=1}^{n} \Pr[A_i] \ = \ n \cdot \left(\frac{2e}{c}\right)^{c \cdot \log n}.
\end{aligned}$$

Again, estimate $(*)$ holds by Boole's inequality. Now, let $c = 4e$. Then, we obtain the following.

$$n \cdot \left(\frac{2e}{c}\right)^{c \cdot \log n} \leq \ n \cdot \left(\frac{1}{2}\right)^{c \cdot \log n} = \ n \cdot \left(\frac{1}{n}\right)^{c} \leq \left(\frac{1}{n}\right)^{\ell}$$

for all $\ell \leq c - 1$.

Case (3)

Given the results above, we can now determine the expected maximum load of the bins $i \in [n]$ of size c_i.

$$\begin{aligned} \mathrm{E}[max_i\{\ell_i\}] &\leq \Pr[\max_i\{\ell_i\} < c \cdot \log n] \cdot c \cdot \log n + \Pr[\max_i\{\ell_i\} \geq c \cdot \log n] \cdot n \log n \\ &\leq \left(1 - \frac{1}{n^\ell}\right) \cdot c \cdot \log n + \frac{n \log n}{n^\ell} \\ &\leq c \cdot \log n + \frac{n \log n}{n^\ell} \leq (c+1) \cdot \log n \in O(\log n). \end{aligned}$$

■

Since the results from Lemma 3.14 hold for any partition $P = \alpha_i \cdot n \log n$ for some constant α_i that is determined by some capacity c_i, the theorem follows. ■

3.1.3 Further Preliminaries

In the following, we sketch some useful mathematical tools taken from probability theory and combinatorial optimization which help to estimate appropriate bounds on the deviation of any disk's load from its expectation.

3.1.3.1 Some Facts from Probability Theory

For our analysis, we use basic results and definitions from fundamental probability theory, like e.g. conditional probability, random variables or expectation. In the following, we introduce some tools which are useful for the analysis of our strategies.

Deviation Bounds. To bound the deviation of the load on a given disk from its expectation, we apply the following version of the prominent *Chernoff Bounds* found in [HR90] that give the *tail estimates based on expectation* for sums of independent binary random variables (we omit the proofs here that are given in detail in e.g. in [MR95]).

Lemma 3.15. (Chernoff Bound) *Consider any set of n independent binary random variables $X_1, \ldots, X_n$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathrm{E}[X]$. Then, it holds for all $\delta \geq 0$ that*

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\min[\delta^2, \delta] \cdot \frac{\mu}{3}}$$

and for all $0 \leq \delta \leq 1$ that

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\delta^2 \cdot \frac{\mu}{2}}.$$

An Independent Bounded Difference Inequality. The following theorem presents tail estimates that are based on bounds on the difference of functions over independent random variables. The given results can be found in [McD89, Sch00, Ros06].

Theorem 3.16. *Let $\mathbf{X} = (X_1, \ldots, X_n)$ be a vector of independent random variables with X_i taking values in a set A_i for each i. Suppose that the real-valued function f defined on $A_1 \times \ldots \times A_n$ satisfies*

$$|f(\mathbf{x}) - f(\mathbf{x}')| \leq c_i$$

whenever the vectors $\mathbf{x}$ and $\mathbf{x}'$ differ only in the ith coordinate. Let μ be the expected value of $f(\mathbf{x})$. Then, for any $d \geq 0$,

$$\Pr[f(\mathbf{X}) - \mu \geq d] \leq e^{-2d^2/\sum_{i=1}^{n} c_i^2}$$

and

$$\Pr[f(\mathbf{X}) - \mu \leq -d] \leq e^{-2d^2/\sum_{i=1}^{n} c_i^2}$$

The Inclusion-Exclusion-Principle on Events. Consider any arbitrary collection of n sets $A_1, \ldots, A_n$. Then, the *inclusion-exclusion principle* is given as

$$\left| \bigcup_{i=1}^{n} A_i \right| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{i_1 < i_2 < \ldots < i_k} \left| \bigcap_{j=1}^{k} A_{ij} \right|.$$

If then $A_1, \ldots, A_n$ is considered as a collection of events, the inclusion-exclusion principle implies

$$\Pr\left[\bigcup_{i=1}^{n} A_i \right] = \sum_{k=1}^{n} (-1)^{k+1} \sum_{i_1 < i_2 < \ldots < i_k} \left[\bigcap_{j=1}^{k} A_{ij} \right]. \tag{3.2}$$

In cases where it is too difficult to evaluate Equation 3.2 exactly, one can use *Boole's inequalities* to find suitable approximations:

$$\Pr\left[\bigcup_{i=1}^{n} A_i \right] \leq \sum_{i=1}^{n} \Pr[A_i]$$

and

$$\Pr\left[\bigcup_{i=1}^{n} A_i \right] \geq \sum_{i=1}^{n} \Pr[A_i] - \sum_{1 \leq i < j \leq n} \Pr[A_i \cap A_j].$$

3.1.3.2 Some Basics from Linear Programming and Farkas' Lemma

Linear programming involves the optimization of a linear function under linear inequality constraints. Since linear programming covers a wide field in mathematics, here, we give only some brief definitions which are useful for the understanding of Farkas' Lemma [Roc70, p.200] (for more details on linear programming, see e.g. [PS98, Ros00])

Definition 3.17. (Basics Facts:) *A **linear programming (LP) problem** is an optimization problem that can be written*

$$\begin{array}{ll} \textit{maximize:} & cx \\ \textit{subject to:} & Ax \geq b \end{array} \tag{3.3}$$

*where $A = (a_{ij})_{i=1,\ldots,q,\, j=1,\ldots,n}$ is a given $q \times n$ matrix, $c \in \mathbb{R}^n$ is a given row vector of length n, and $b \in \mathbb{R}^q$ is a given column vector of length q (note that in (3.3), the '$\geq$'-relation is defined on vectors, i.e. for each component in the vectors). Each of the q inequalities in $Ax \geq b$ is a **constraint**, and the **decision variables** of the Problem 3.3 are represented by the column vector x of length n.*

*A **feasible solution** is a vector x satisfying $Ax \geq b$. The **feasible region** is the subset of all feasible solutions in $\mathbb{R}^n$. If no feasible solution exists (so that the feasible region is empty), the LP problem is **infeasible**; otherwise it is **feasible**.*

*For a feasible solution x, the function $z = cx$ is the **objective function** and cx the **objective value** of x. If the feasible solution is also the maximum, the feasible solution x^* is an **optimal** solution. If the optimal value can be made arbitrarily large over the feasible region, the LP problem is **unbounded**. Otherwise it is **bounded**.*

*Since every LP problem, referred to as a **primal problem**, can be converted into a **dual problem** providing an upper bound to the optimal value of the primal problem, and additionally, every feasible solution for an LP gives a bound on the optimal value of the objective function of its dual. The **weak duality theorem** states that the objective function value of the dual at any feasible solution is always greater than or equal to the objective function value of the primal at any feasible solution. The **strong duality theorem** states that if the primal has an optimal solution, x^*, then the dual also has an optimal solution, y^*, such that $c^T x^* = b^T y^*$.*

The duality theorems given above base on the following *Farkas' Lemma* which is an example of a *theorem of the alternative*; a theorem stating that of two systems, one or the other has a solution, but not both. Thus, it can be applied to show the existence (and uniqueness) of solutions to linear models and stationary distributions in finite Markov chains.

Lemma 3.18. (Farkas' Lemma) *Let A be a real-valued $q \times n$ matrix and x and b are vectors. Then, the system:*

$$Ax = b,\ x \geq 0$$

has no solution if and only if the system

$$A^T y \geq 0,\ b^T y < 0$$

has a solution, where y is a vector of length q.

3.1.3.3 Hypergraphs

The analysis in the next section depends on structures described by hypergraphs for which we give the major properties we require by the following definition.

Definition 3.19. (Hypergraphs) *Let V be a finite set of **nodes**. Further, let $E \subseteq \bigcup_{i \geq 2} V^i$ be a finite subset of the set of tuples over V. The set E is called the set of **edges**. An element $e = (v_1, \ldots, v_j) \in E$ is said to be **self-loop-free** if $v_i \neq v_{i'}$ for all $1 \leq i \leq i' \leq j$. If all elements are self-loop-free, the pair $G = (V, E)$ is called a **hypergraph**. A hypergraph $G = (v, E)$ is **d-uniform** if $E \subseteq V^d$. An edge $e \in E$ has **size** $\mathbf{d}$ if $e \in V^d$.*

*Let $e = (v_1, \ldots, v_j) \in E$ be an edge of a hypergraph G and let $v \in V$ be a node of G. The node v is said to be **incident** to e if there is an $1 \leq i \leq j$ with $v_i = v$. An edge e is **incident** to a node v if v is **incident** to e. If a node v is incident to a number i of edges, the node v has **degree i** ($deg(v) = i$). If $v \neq v'$ are nodes incident to the same edge $e \in E$, the nodes v and v' are said to be **adjacent**.*

*For $j \geq 1$, a sequence of nodes $v_1, \ldots, v_j$ is called a **path** connecting v_1 and v_j if v_i is adjacent to v_{i+1} for each $1 \leq i \leq j-1$. The **length** of a path $v_1, \ldots, v_j$ is $j-1$. Two nodes v and w are said to have **distance i** ($dist(v, w) = i$) if the shortest path connecting v and w has length i. If $W \subseteq V$ is a subset of the nodes, the **distance** between a node $v \in V$ and W is defined by $min_{w \in W}\ dist(v, w)$. Two paths $v_1, \ldots, v_i$ and $w_1, \ldots, w_j$ are **distinct** if $\{v_1, \ldots, v_i\} \cap \{w_1, \ldots, w_j\} = \emptyset$.*

3.1.3.4 Multisets

The following definition of multisets is well-known from set theory.

Definition 3.20. *A **multiset** $M(E)$ is a pair $(E, \#_E)$ where E is some **underlying set** and $\#_E : E \to \mathbb{N}_0$ is a function specifying the multiplicity of the elements in E. For each $e \in E$, let $\mu_E(e) := \#_E(e)$ denote its multiplicity. Thus, $\mu_E(e) = 0$ for all $e \notin E$. Furthermore, the **cardinality** of a multiset $|M(E)| = \sum_{e \in E} \mu_E(e)$.*

***Subtraction** on multisets is defined as $M(E') = M(E) \setminus e$, thus, $\mu_{E'}(e) := \mu_E(e) - 1$ if $\mu_E(e) > 0$ and $\mu_{E'}(e) := \mu_E(e) = 0$ otherwise. The **union of multisets** is defined analogously, i.e. $\mu_{E'}(e) := \mu_E(e) + 1$.*

3.1.3.5 Competitive Analysis

In part two of this chapter, we analyze the introduced strategies by applying *competitive analysis* which is a well-known method of evaluating *online algorithms* (see [BEY98]).

With respect to our dynamic data placement strategies, we apply competitive analysis to evaluate their adaptivity. That is, for any change operation ω, we compare the number of (re)-placements of data blocks performed by the observed strategy with the number of (re)-placements performed by an optimal strategy such that all required conditions are satisfied. To formalize our intension, we introduce the following definition.

Definition 3.21. (Competitiveness) *Let $S(n,C[n])$ be a storage network, and let k be the identifier of some storage device. Then, let $L = \{insert(S(n,C[n]),k), delete(S(n,C[n]),k)\}$ be the set of operations on $S(n,C[n])$ that cause changes in the number of connected storage devices. Furthermore, let A denote a dynamic data placement strategy on $S(n,C[n])$.*

*Then, for any operation $\omega \in L$, the **competitive ratio $comp_A(\omega)$ of A on ω** is the number*

$$comp_A(\omega) = \max\left\{\frac{C_{\mathrm{Opt}}(\omega)}{C_A(\omega)}, \frac{C_A(\omega)}{C_{\mathrm{Opt}}(\omega)}\right\},$$

where $C_{\mathrm{Opt}}(\omega)$ denotes the cost of an optimal algorithm for the instance ω, and $C_A(\omega)$ denotes the cost of the online strategy A.

*Let $c \geq 1$. Then, the storage strategy A is called **c-competitive** concerning operation ω if, on expectation, $comp_A(\omega) \leq c$ for all $\omega \in L$.*

Note that randomized algorithms must fulfill this bound with high probability, that is, the number of redistributed data blocks implied by an operation ω should be at most $\ell \cdot (c \cdot C_{\mathrm{Opt}}(\omega))$ with probability at least $1 - \left(\frac{1}{n}\right)^{\ell}$.

3.2 The Redundant Allocation Problem

Given all the formal definitions and required mathematical tools, we can now refine the redundant allocation problem with respect to a rather probabilistic based formulation. In particular, the following *central problem of allocation* describes the major allocation problem underlying the design and analysis of all schemes we discuss in this chapter. In other words, all algorithms we introduce in the following, whether adaptive or not, basically aim at finding a suitable solution for this allocation problem.

> **Central Problem of Allocation:** *Given a redundant allocation problem for a given storage network $S(n,C[n])$, find an appropriate probability distribution P on the set $[n]$ of disks in either a static or dynamic setting that satisfies the redundancy and fairness conditions for any number $m \leq C_{total}$ of data blocks.*

Clearly, if we ignore redundancy, we can simply set $P = \bar{c}$ because then, all blocks are allocated independently at random according to P. For such settings, there exist numerous strategies ensuring a fair allocation for static as well as for dynamic storage systems (note that we omit dynamic systems in this section but return back to them in Section 3.5). For a static setting, a fair data layout can easily be achieved by applying standard *balls-into-bins models*. Again, given a storage network $S(n, C[n])$ and $P = \bar{c}$, a balls-into-bins allocation stores each data item (ball) $d \in [k]$ independently at random to some bin $i \in [n]$ with probability c_i. For non-uniform capacities, this yields a fair data layout up to some factor $(1 \pm \varepsilon)$, whereas in a uniform setting, we can come close to optimal if one out of $d \geq 2$ choices for each allocation is allowed (we return to standard balls-into-bins games a little more precisely in Section 3.3.2).

However, whenever regarding redundancy, the observed problem becomes more challenging because, as we know from the realization of the function F in Section 3.1.2, the redundancy condition imposes high carefulness on the allocation functions, and more importantly, for any r identical copies, $r-1$ allocations are no longer independent. This implies a process in which for each of the functions $f_1, f_2, \ldots, f_r$ in F, respectively for a given sequence of r identical copies, we always have to find a sequence $P = (P_1, P_2, \ldots, P_r)$ of probability distributions, where P_i is valid for all allocations via function f_i only.

Therefore, in what follows in this section, we concentrate on the investigation of appropriate sequences $P = (P_1, P_2, \ldots, P_r)$ subject to a fair data allocation. At first, we describe a very intuitive balls-into-bins-like algorithm which aims at ensuring a redundant allocation on the share tuples of disks that remain valid in some particular allocation step (note that for a data item $d \in [k]$ and any step $i \in [r]$, the random allocation of the i-th copy of d to some disk corresponds to the allocation $f_i(d)$ of F).

Algorithm 1 Naive Balls-into-Bins Allocation

Require: Probability Space $(S(n, C[n]), \bar{c})$, Set $[k]$ of balls, r-tuple $(f_1, f_2, \ldots, f_r)$
Ensure: Redundant Allocation of $m = k \cdot r$ replicated balls

1: **for all** $d \in [k]$ **do**
2: To hold the r-th copy of d, choose disk $f_r(d) = i_r \in [n]$ with probability c_{i_r}
3: **repeat**
4: Set $[n'] \leftarrow ([n] \setminus \{i_r\})$ {Reduce sample space by chosen disk i_r}
5: Set $r' \leftarrow (r-1)$ {Reduce current number of copy}
6: **for all** $i \in [n']$ **do**
7: Set $c'_i \leftarrow (C_i / \sum_{j \in [n']} C_j)$ {(Re-)assign disk probabilities (relative capacities)}
8: To hold the r'-th copy of d, choose disk $f_{r'}(d) = i_{r'} \in [n']$ with probability $c'_{i_{r'}}$
9: **until** $(r' = 0)$

Obviously, this approach is quite self-evident for obtaining a fair allocation under redundancy conditions because it is simple, features low complexity and is closely related to

the well-analyzed standard balls-into-bins model for independent allocations. However, as we show in the following, the given algorithm is only useful for uniform capacities because for heterogeneous settings, a naive balls-into-bins scheme always induces an imbalance on the data layout such that fairness can never be obtained. Since, generally, balls-into-bins schemes feature high attraction to many designers of data placement algorithms, it is worth investigating the occurring imbalance in more detail. We start by first examining a homogeneous setting for Algorithm 1.

Theorem 3.22. *Let $S(n,C[n])$ be a uniform storage network with relative capacities $c_i = \frac{1}{n}$ for all disks $i \in [n]$. Furthermore, let $[k]$ be a set of data items and $r \in [n]$ the replication parameter for each $d \in [k]$. Then, for any number $m \leq C_{total}$ ($m = k \cdot r$) of replicated blocks, Algorithm 1 realizes a balanced (i.e. equal) load on the set $[n]$ of disks.*

Proof. In fact, this process can be seen as (the first r steps of) sampling a permutation $\pi : [n] \to [n]$ uniformly at random: first choose $\pi(1) \in [n]$ (with probability $\frac{1}{n}$), then $\pi(2) \in [n]' = [n] \setminus \{\pi(1)\}$ and so on. Since for each $i \in [r]$, $\pi(i)$ is chosen uniformly from $[n] \setminus \{\pi(1), \ldots, \pi(i-1)\}$, that is, with probability $\frac{1}{n-i+1}$ independent of the values $\pi(1), \ldots, \pi(i-1)$, induction on r shows that each r-tuple $\big(\pi(1), \ldots, \pi(r)\big)$ of pairwise distinct numbers between 1 and n appears with equal probability $\frac{n-r!}{n!}$. Proceeding from an r-tuple $\big(\pi(1), \ldots, \pi(r)\big)$ to the r-element set $\{\pi(1), \ldots, \pi(r)\}$ constitutes a mapping with exactly $r!$ fibers of equal probability $\frac{n-r!}{n!}$ each: every r-element subset therefore arises with probability $1/\binom{n}{r}$. ∎

Observation 3.23. *Given a uniform storage network $S(n,C[n])$, Theorem 3.22 constitutes an efficient algorithm for sampling uniformly at random an r-element subset of the set $[n]$ of disks. For the subsequent descriptions, let us interpret Algorithm 1 differently: consider the uniform sampling of a hyperedge E from the complete r-uniform $\binom{n-1}{r-1}$-regular hypergraph $\binom{[n]}{r} := \{E \subseteq [n] \mid |E| = r\}$. Notice that this random hyperedge E contains any fixed disk $i \in [n]$ with probability $p_i = \frac{r}{n}$ (because we have n different elements, each of which chosen with probability $\frac{1}{n}$ and r possible entries for each of the elements in $[n]$).*

So far the homogeneous case. Now consider Algorithm 1 for the inhomogeneous case. For this, in the following, we restrict ourselves to random processes which are *online* and *memory-less* in the sense that they distribute (the r copies of) each given data block independently from the previous ones. Thus, balanced load then means that the r disks which are drawn for storing the copies of some current data block should be sampled from the set $[n]$ in a way such that disk i gets chosen with probability $p_i := \sum_{E \in \binom{[n]}{r} : i \in E} \Pr[E]$, proportional to its relative capacity $c_i = C_i / \sum_j C_j$.

However, as the following example shows, this assignment (and thus Algorithm 1) fails for non-uniform capacities.

Example 3.24. *For simplicity, in the following two examples, we consider the size of the disks due to the normalized data items of size 1.*

a) Consider the case $n = 3$, $r = 2$, and disks $1,2,3$ with capacities $C_1 = C_2 = 1$, $C_3 = 2$.

In order to fill these disks in a balanced way, disk 3 has to receive twice as many copies as both disk 1 and 2. That is, to obtain fairness (and thus, full utilization), disk tuples $\{1,3\}$ and $\{2,3\}$ must be chosen as where to place copies with probability $\frac{1}{2}$ each; whereas tuple $\{1,2\}$ must not occur. In contrast, Algorithm 1 chooses disk 1 with probability $c_1 = \frac{1}{4}$ to hold the first copy; and then disk 2 with probability $c'_2 = \frac{1}{3}$ yielding a positive chance for the tuple $\{1,2\}$ to occur.

b) Even worse in case $n = 3$, $r = 2$, and $C_1 = C_2 = 1$, $C_3 = 3$, there is no way to achieve a balanced distribution at all, regardless of which algorithm one employs !

According to the given translation on hypergraphs, we can now refine our examinations on finding a suitable probability distribution P for the disjoint storage of replicated balls. We start with posing the following central question.

Question 3.25. *Consider the replication parameter r and fixed integers $p_1, \ldots, p_n \in [0,1]$.*

a) Does there exist a probability distribution $P : \binom{[n]}{r} \to [0,1]$ such that[6]

$$\forall i \in [n] : \qquad p_i \quad = \sum_{E \in \binom{[n]}{r} : i \in E} P(E) \ ? \tag{3.4}$$

b) Is such a distribution uniquely determined by Equation 3.4 ?

c) How can one efficiently algorithmically determine this distribution P from the tuple $\bar{p} = (p_1, \ldots, p_n)$?

d) How can one efficiently algorithmically sample $E \in \binom{[n]}{r}$ according to P for either static or dynamic systems ?

For the rest of this chapter, we concentrate on answering the questions above. In particular, in the following section, we examine the properties of a possibly existing distribution P in more detail. Section 3.5 then is dedicated to the latter two questions above.

3.3 Analysis of the Probability Distribution P

Based on the above observations, we now analyze the properties of P in terms of existence, uniqueness, and of algorithmic tractability. Moreover, we show that using a balls-into-bins scheme for allocation as realized by Algorithm 1 always yields an imbalance on the resulting data layout whenever replicated balls are to be stored on heterogeneous disks.

[6] To emphasize the probability distribution P, in this section, we use $P(E)$ rather than $\Pr[E]$ to denote the probability of some event E.

3.3.1 Existence and Uniqueness

Regarding Question 3.25(a), a simple double-counting gives rise to the following necessary (pre-)condition:

Lemma 3.26. *Consider a storage network $S(n,C[n])$ and fixed replication parameter r. Then, $P:\binom{[n]}{r}\to[0,1]$ denotes a probability distribution on the set $[n]$ of disks such that $p_i := \sum_{E\in\binom{[n]}{r}:i\in E} P(E)$ satisfies $\sum_{i=1}^{n} p_i = r$.*

Proof.

$$\sum_{i=1}^{n} p_i \;=\; \sum_{(i,E)\in[n]\times\binom{[n]}{r}:i\in E} P(E) \;=\; \sum_{E\in\binom{[n]}{r}} \underbrace{\sum_{i\in E} P(E)}_{=P(E)\cdot|E|} \;=\; r\cdot \sum_{E\in\binom{[n]}{r}} P(E) = r \tag{3.5}$$

■

Note that subject to Definition 3.5, we require $p_i = r\cdot c_i$ later for our allocations. Then, to achieve a fair and redundant data distribution **at all**, Lemma 3.26 implies that for the relative capacities c_i of all disks $i\in[n]$, it must hold that $c_i \le \frac{1}{r}$ (c.f. Lemma 3.29).

3.3.1.1 Existence

Given the result from Lemma 3.26, we can now focus on the existence of the probability distribution P regarding the above condition.

Theorem 3.27. *Let $p_1,\ldots,p_n\in[0,1]$ such that $\sum_i p_i = r$. Then, there exists a probability distribution $P:\binom{[n]}{r}\to[0,1]$ with $p_i = \sum_{E\in\binom{[n]}{r}:i\in E} P(E)$ for each $i\in[n]$.*

Proof. Consider the incidence matrix A of the r-uniform $\binom{n-1}{r-1}$-regular hypergraph $\binom{[n]}{r}$, that is, $A=(a_{i,E})_{(i,E)\in[n]\times\binom{[n]}{r}}$ where $a_{i,E}=1$ if $i\in E$ and $a_{i,E}=0$ otherwise. We are thus looking for a vector

$$x=\big(P(E)\big)_{E\in\binom{[n]}{r}}\in[0,\infty)^{\binom{[n]}{r}} \quad \text{s.t.} \quad A\cdot x=\Big(\sum_{E\in\binom{[n]}{r}:i\in E} x_E\Big)_{i\in[n]} \stackrel{!}{=} (p_i)_{i\in[n]} \;. \tag{3.6}$$

Let A^T denote the transposed matrix of A. Then, we can observe that for vector $y\in\mathbb{R}^n$, $A^T\cdot y$ is a vector of size $\binom{n}{r}$ with E-th component equal to $\sum_{i\in E} y_i$ where $E\in\binom{[n]}{r}$. Now, $A^T\cdot y\ge 0$ is infeasible for $\sum_i y_i p_i<0$ according to the claim below; hence by Farkas' Lemma ([Roc70]), we obtain that Equation 3.6 is feasible. Moreover, reading Equation 3.5 reversely reveals such a solution x to satisfy $\sum_{E\in\binom{[n]}{r}} x_E = 1$, i.e. to yield a probability distribution P. ■

Lemma 3.28. *Let $y_1, y_2, \ldots, y_n \in \mathbb{R}$ and $p_1, \ldots, p_n \in [0,1]$ with $\sum_i p_i = r \in \mathbb{N}$. Suppose that $\sum_{i \in E} y_i \geq 0$ for each $E \in \binom{[n]}{r}$. Then, it holds that $\sum_{i=1}^{n} y_i p_i \geq 0$.*

Proof. W.l.o.g. let $y_1 \leq y_2 \leq \cdots \leq y_n$. Then, $\sum_{i=1}^{n} y_i p_i$ becomes minimal (subject to the constraints $0 \leq p_1, \ldots, p_n \leq 1$ and $\sum_i p_i = r$) for $1 = p_1 = \cdots = p_r$ and $0 = p_{r+1} = \cdots = p_n$. But that minimum equals $\sum_{i \in E} y_i \geq 0$ for $E = \{1, \ldots, r\}$. ■

So far, we have shown that there exists a probability distribution P as asked for in Question 3.25(a). Now, we tackle on the uniqueness of such distribution, or in other words, are there more than one possibilities to assign probabilities to the disks in the set $[n]$ in order to obtain the desired fair and redundant allocation ?

3.3.1.2 Uniqueness

Uniqueness of P would mean that $\bar{p} = (p_1, \ldots, p_n)$ determines $P : \binom{[n]}{r} \to [0,1]$ subject to Condition 3.4. However, this fails already in the uniform case on $\binom{[4]}{2}$, that is, with $p_i = \frac{1}{2}$ for all $i \in [4]$: In addition to the straight-forward solution $P = \frac{1}{6}$, also the following one is easily verified to satisfy Equation 3.4.

$$\begin{aligned}
P(\{1,2\}) &= \tfrac{1}{6} - \tfrac{1}{11}, & P(\{1,3\}) &= \tfrac{1}{6} - \tfrac{1}{7}, & P(\{1,4\}) &= \tfrac{1}{6} + \tfrac{1}{11} + \tfrac{1}{7}, \\
P(\{2,3\}) &= \tfrac{1}{6} + \tfrac{1}{11} + \tfrac{1}{7}, & P(\{2,4\}) &= \tfrac{1}{6} - \tfrac{1}{7}, & P(\{3,4\}) &= \tfrac{1}{6} - \tfrac{1}{11} \quad .
\end{aligned}$$

The given example proves that Equation 3.4 fails to uniquely determine a probability distribution P for our redundant allocations. Thus, consequently, there is not only a single chance to find the desired data layout.

In particular, the most intuitive way of determining a correct P obviously would be to turn the proof of Theorem 3.27 into an algorithm as follows: given a set $[n]$ of bins with relative capacities $c_1, c_2, \ldots, c_n \in [0,1]$, we define $p_i := r \cdot c_i$ satisfying $\sum_i p_i = r$ (Lemma 3.26). Then, one could use his favorite linear optimizer to solve Equation 3.6. Since linear programming is in the complexity class P (see e.g. [GLS93]), this yields a valid probability distribution $P : \binom{[n]}{r} \to [0,1]$ as desired using running time polynomial in $\binom{[n]}{r} = \Theta\left(\frac{n}{r}\right)^r$.

However, merely storing P completely may already take exponential space (for instance, for $r = \frac{n}{2}$). Thus, obviously, one would be better off just sampling some hyperedge $E \in \binom{[n]}{r}$ according to the distribution P as this can be performed without actually calculating P entirely.

Therefore, with respect to the very efficient and successful allocations gained by standard balls-into-bins games for the non-redundant case, it seems reasonable to apply Algorithm 1 also for redundant settings. Some further motivating fact is that in Theorem 3.22, we already proved that for a uniform configuration with $c_1 = c_2 = \ldots = c_n = \frac{1}{n}$, the depicted Algorithm 1 provides the desired data layout.

3.3.2 On the Imbalance Induced by Balls-into-Bins Allocations

Before we start comparing the results and properties of Algorithm 1 with a standard balls-into-bins model, we first give some brief overview about the most important results on allocating data blocks by a balls-into-bins approach. Notice that since there are no useful results considering redundant placement, we only refer to independent allocations.

3.3.2.1 Previous Results on Balls-into-Bins

In a standard balls-into-bins model, one sequentially allocates a set of m independent balls at random to a set of n bins such that the maximum number of balls in any bin is minimized. This paradigm has become very popular because it is simple, quite general, and can be adopted for a variety of applications. To our knowledge, the first use was given by Karp et al. [KLM92] who deal with PRAM simulations on Distributed Memory Machines (DMMs). They realize the balls-into-bins paradigm by universal hash functions that map the PRAM memory cells to the memory modules of the DMM in a way that an even dispersal of the cells over the DMM modules can be obtained such that, for any access request to some PRAM cell, an efficient response time can be ensured.

Depending on the number of used hash functions, the mapping of the cells to the memory modules is either realizing a so-called single-choice or multiple-choice approach. The classical *single-choice approach* allocates each ball to a bin chosen independently and uniformly at random. For $m = n$ balls, the maximum load in any bin is $\Theta\left(\frac{\log n}{\log\log n}\right)$. More generally, for $m \geq n \log n$ balls, it is well-known that there exists a bin receiving $\frac{m}{n} + \Theta\left(\sqrt{\frac{m \log n}{n}}\right)$ balls (see e.g. [RS98]). This result holds not only on expectation but with high probability. Let the *maximum height above average* denote the difference between the number of balls in the fullest bin and the average number of balls per bin. Then, the maximum height above average of the single-choice approach is $\Theta\left(\sqrt{\frac{m \log n}{n}}\right)$, and it is easy to see that the deviation between the randomized single-choice allocation and the optimal allocation increases with the number of balls. Importantly, this cannot be done better for this kind of game, i.e. for single-choice processes.

Nevertheless, the maximum load can be decreased dramatically by allowing every ball to choose one out of $d \geq 2$ bins independently and uniformly at random. The classical results for this *multiple-choice game* are given by Azar et al. [ABKU00] who, following the results from the PRAM simulations, introduced the Greedy[d] protocol for allocating n balls to n bins. In each step, Greedy[d] chooses $d \geq 2$ bins for each ball and allocates the ball into a bin with minimum load. Then, it can be shown that the maximum height above average obtained by Greedy[d] is only $\frac{\log\log n}{\log d} + \Theta\left(\frac{m}{n}\right)$, w.h.p. which is an *exponential decrease* compared to single-choice games. Moreover, Vöcking [Voe99] introduced the Always-Go-Left protocol yielding a maximum height above average of only

$\frac{\log\log n}{\log d^*} + \Theta\left(\frac{m}{n}\right)$ with $d^* \approx d \log 2$. Since all these results hold only for the case $m \approx n$, in [BCSV00], the authors analyze Greedy[d] for the *heavily loaded case*, i.e. $m \gg n$ (what is, clearly, a more suitable investigation when considering the allocation of data blocks to storage devices). They show that the maximum load is $\frac{m}{n} + \frac{\log\log n}{\log d}$, w.h.p. This result yields a fundamental difference between the allocation obtained by multiple- and the single-choice approach; while the single-choice allocation deviates more and more from the optimal allocation with an increasing number of balls, the deviation gained by the multiple-choice allocation is independent from the number of balls.

The above results show that with a multiple-choice allocation, on one hand, a near-optimal load balancing can be obtained while, on the other hand, the analysis of these allocations is challenging even for uniform capacities and pairwise independent balls. With respect to heterogeneous bins, Wieder [Wie07] introduced the analysis of a multiple-choice allocation obtaining a tight upper bound of $\frac{m}{n} + \frac{\log\log n}{\log(1+\varepsilon)} + O(1)$ w.h.p. using a modification of the Greedy[d] algorithm called Greedy[$U, 1+\varepsilon$], where U denotes the uniform probability distribution on n bins. Greedy[$U, 1+\varepsilon$] inserts m balls sequentially, and in each round, the probability of a ball to be inserted into one of the i-th most heavy bins is exactly $(\frac{i}{n})^{1+\varepsilon}$. However, two main drawbacks come along with this approach. First, Greedy[$U, 1+\varepsilon$] is not of practical usage, and second, the analysis is derived from allocations in peer-to-peer networks, that is, the author assumes homogeneous peers (all of same size) to which an imbalanced probability distribution is wrongfully assigned (the imbalance is caused by a previous allocation process, like e.g. CHORD [SMK+01a], such that an appropriate uniform distribution on the peers is biased). Obviously, this scenario does not reflect the assumptions and demands of the system we observe here.

3.3.2.2 Analysis of the Imbalance

In Lemma 3.22, we showed that symmetry considerations significantly simplified the analysis of Algorithm 1 in a uniform system because in such settings, the dependencies between any two choices of disks are always the same. This implies that every r-tuple $\big(\pi(1),\ldots,\pi(r)\big)$ was equally likely to be chosen for allocation, which in turn implies that also all their $r!$ fibers $\{\pi(1),\ldots,\pi(r)\}$ have equal probability. In contrast, as we show in this section, this behaves significantly different if any non-uniform capacity distribution is considered. In particular, Theorem 3.30 shows that the referred algorithm, realizing the redundant type of a standard balls-into-bins game, fails when being applied in heterogeneous storage environments for which an optimal disk utilization is required.

As usual, we consider a given storage network $S(n, C[n])$, and since all of our strategies work on values defined by the share tuples $\bar{c}$ for the disks $[n]$, before we start, we first state an important precondition on the capacity of each disk which is necessary for obtaining a fair and redundant data layout at all.

Lemma 3.29. *A fair and redundant distribution is feasible if and only if $c_i \leq \frac{1}{r}$ for all bins $i \in [n]$.*

Proof. Consider the random variables from Definition 3.5. Now, assume some bin j with relative capacity $c_j > \frac{1}{r}$. Then, for the variable X_j, it holds that $\mathrm{E}[X_j] = r \cdot k \cdot c_j > r \cdot k \cdot \frac{1}{r}$, thus, bin j contains at least two identical copies violating the redundancy condition. ■

However, even by supposing that all disks i have relative capacity $c_i \leq \frac{1}{r}$, the following theorem shows that Lemma 3.29 states only a necessary condition on the allocation.

Theorem 3.30. *Consider a storage network $S(n, C[n])$ and a share tuple $\bar{c}$ with $c_i \leq \frac{1}{r}$ for all $i \in [n]$. If then there exists at least any two indices $i, j \in [n]$ with $c_i \neq c_j$, Algorithm 1 never provides a fair allocation on the set $[n]$ of disks.*

Proof. Consider the share tuple $\bar{c} = \{c_1, c_2, \ldots, c_n\}$ and the replication parameter r. Moreover, for non-uniformity and w.l.o.g., we assume $c_1 < c_n$. Then, we start by defining the values $p_i := r \cdot c_i$, and since $\sum_{i=1}^{n} c_i = 1$, it holds that $\sum_{i=1}^{n} p_i = r$ satisfying the necessary conditions for balanced allocations in Lemma 3.26. For what follows, we further assume w.l.o.g. that $p_1 \leq p_2 \leq \ldots \leq p_n$, and we define the following *extracting function*

$$\Xi := (\xi_1, \xi_2, \ldots, \xi_r) : \binom{[n]}{r} \to [n]^r$$

covering r component functions $\xi_1, \xi_2, \ldots, \xi_r$ with

$$\xi_i \ : \ \binom{[n]}{r} \to [n] \quad \text{s.t. the condition} \quad \bigcup_{i=1}^{r} \{\xi_i(E)\} = E \quad \forall E \in \binom{[n]}{r}$$

on the considered hypergraph which, given some edge $E \in \binom{[n]}{r}$, extracts an r-tuple by the component functions. More precisely, each function ξ_i extracts an element from E, and according to its position i in Ξ, ξ_i thus determines the i-th element of the resulting r-tuple. Moreover, as due to the referred condition all extracted integers are pairwise distinct, the functions define an order on the elements in E.

Now, consider the symmetric group S_r on the set $[r]$. Then, for all permutations $\pi \in S_r$ and any edge $E \in \binom{[n]}{r}$, we define

$$\Xi_\pi(E) := (\xi_{\pi(1)}(E), \xi_{\pi(2)}(E), \ldots, \xi_{\pi(r)}(E)).$$

W.l.o.g. let $\xi_r(E) = n$ for some edge $E \in \binom{[n]}{r} : n \in E$. Then, for the rest of this proof, we aim at stating that

$$p_n \ > \sum_{E \in \binom{[n]}{r} : n \in E} P(E) \tag{3.7}$$

violating Theorem 3.27 and thus showing that fairness can not be achieved.

Since we are focusing on a balls-into-bins game, for each r-tuple of identical copies, we consider a sequential algorithm to place the r copies to the set $[n]$ of disks. Thus, for any r-tuple and any bin $i \in [n]$, let E_i^t denote the event that bin i is drawn for storing the t-th copy, $1 \leq t \leq r$. Moreover, according to the redundancy condition, only the first copy of each tuple is allocated independently from any other placement. Hence, for redundantly allocating any number $\ell \leq r$ of identical copies to bins $i_1, i_2, \ldots, i_\ell$, let the conditional probabilities $q_{i_1,i_2,\ldots,i_\ell} := \Pr[E_{i_1}^1 \cap E_{i_2}^2 \cap \ldots \cap E_{i_\ell}^\ell]$ be defined as

$$q_{i_1,i_2,\ldots,i_\ell} := c_{i_1} \cdot \frac{c_{i_2}}{1-c_{i_1}} \cdot \ldots \cdot \frac{c_{i_\ell}}{1-c_{i_1}-c_{i_2}\ldots-c_{i_{\ell-1}}} \tag{3.8}$$

Now, consider the allocation of $\ell+1$ identical copies. Then, for the probability $q_{i_1,i_2,\ldots,i_\ell}$, we can state the following lemma.

Lemma 3.31.

$$q_{i_1,i_2,\ldots,i_\ell} = \sum_{\substack{i_{\ell+1}\in[n] \\ i_{\ell+1}\neq i_1,i_2,\ldots,i_\ell}} q_{i_1,i_2,\ldots,i_\ell,i_{\ell+1}}$$

Proof. Since $\sum_{i=1}^n c_i = 1$, we can state:

$$\begin{aligned}
& \sum_{\substack{i_{\ell+1}\in[n] \\ i_{\ell+1}\neq i_1,i_2,\ldots,i_\ell}} q_{i_1,i_2,\ldots,i_\ell,i_{\ell+1}} \\
= \; & c_{i_1} \cdot \frac{c_{i_2}}{1-c_{i_1}} \cdot \ldots \cdot \frac{c_{i_\ell}}{1-c_{i_1}-c_{i_2}\ldots-c_{i_{\ell-1}}} \cdot \frac{\sum_{\substack{i_{\ell+1}\in[n] \\ i_{\ell+1}\neq i_1,i_2,\ldots,i_\ell}} c_{i_{\ell+1}}}{1-c_{i_1}-c_{i_2}-\ldots-c_{i_{\ell-1}}-c_{i_\ell}} \\
= \; & c_{i_1} \cdot \frac{c_{i_2}}{1-c_{i_1}} \cdot \ldots \cdot \frac{c_{i_\ell}}{1-c_{i_1}-c_{i_2}\ldots-c_{i_{\ell-1}}} \cdot \frac{1-c_{i_1}-c_{i_2}-\ldots-c_{i_{\ell-1}}-c_{i_\ell}}{1-c_{i_1}-c_{i_2}-\ldots-c_{i_{\ell-1}}-c_{i_\ell}} \\
= \; & q_{i_1,i_2,\ldots,i_\ell}
\end{aligned}$$

■

Now, returning to Inequality 3.7, we obtain

$$\sum_{E\in\binom{[n]}{r}:n\in E} P(E) \;=\; \sum_{E\in\binom{[n]}{r}:n\in E} \sum_{\pi\in S_r} P[(\xi_{\pi(1)}(E), \xi_{\pi(2)}(E), \ldots, \xi_{\pi(r)}(E))].$$

Due to Definition 3.8 above, let $q_{\pi(1),\pi(2),\ldots,\pi(r)}(E) := q_{\xi_{\pi(1)}(E),\xi_{\pi(2)}(E),\ldots,\xi_{\pi(r)}(E)}$. Then,

$$\sum_{E\in\binom{[n]}{r}:n\in E} \sum_{\pi\in S_r} P[(\xi_{\pi(1)}(E), \ldots, \xi_{\pi(r)}(E))] \;=\; \sum_{E\in\binom{[n]}{r}:n\in E} \sum_{\pi\in S_r} q_{\pi(1),\ldots,\pi(r)}(E)$$

Now, for all $E \in \binom{[n]}{r} : n \in E$ and indices $i \in [r]$, let us consider the cases with bin n at position $\xi_{\pi(i)}$. Then,

$$\begin{aligned}
&\sum_{E\in\binom{[n]}{r}:n\in E} \sum_{\pi\in S_r} q_{\pi(1),\ldots,\pi(r)}(E) \\
= &\sum_{E\in\binom{[n]}{r}:n\in E} \left(\sum_{\substack{\pi\in S_r \\ \xi_{\pi(1)}(E)=n}} q_{\pi(1),\pi(2),\ldots,\pi(r)}(E) + \ldots + \sum_{\substack{\pi\in S_r \\ \xi_{\pi(r)}(E)=n}} q_{\pi(1),\pi(2),\ldots,\pi(r)}(E) \right) \\
= &\sum_{E\in\binom{[n]}{r}:n\in E} \sum_{\substack{\pi\in S_r \\ \xi_{\pi(1)}(E)=n}} q_{\pi(1),\pi(2),\ldots,\pi(r)}(E) + \ldots + \sum_{E\in\binom{[n]}{r}:n\in E} \sum_{\substack{\pi\in S_r \\ \xi_{\pi(r)}(E)=n}} q_{\pi(1),\pi(2),\ldots,\pi(r)}(E)
\end{aligned}$$

Before we continue our descriptions, we first take a closer look at the summands given in the above summation. In particular, for some index k, $1 \leq k \leq r$, let us fix the case for which $\xi_{\pi(k)} = n$. Additionally, for the sake of convenience, we define the following sets of tuples of indices

$$I_\ell^u := \{(i_\ell, i_{\ell+1}, \ldots, i_u) \in [n]^{u-\ell+1} \mid i_a \neq i_b; \forall l \leq a, b \leq u : a \neq b\} \tag{3.9}$$

Then,

$$\begin{aligned}
&\sum_{E\in\binom{[n]}{r}:n\in E} \sum_{\substack{\pi\in S_r \\ \xi_{\pi(k)}(E)=n}} q_{\pi(1),\pi(2),\ldots,\pi(r)}(E) \\
= &\sum_{\substack{(i_1,i_2,\ldots,i_r)\in I_1^r \\ i_k=n}} q_{i_1,i_2,\ldots,i_r} \\
= &\sum_{(i_1,i_2,\ldots,i_{k-1})\in I_1^{k-1}} \sum_{(i_{k+1},\ldots,i_r)\in I_{k+1}^r} q_{i_1,i_2,\ldots,i_{k-1},n,i_{k+1},\ldots,i_r} \\
= &\sum_{(i_1,i_2,\ldots,i_{k-1})\in I_1^{k-1}} q_{i_1,i_2,\ldots,i_{k-1},n}
\end{aligned}$$

which follows from Lemma 3.31 by induction. More precisely, since we concentrate on the allocation on bin n, we can simply ignore the last $r-k$ steps of the r-fold placement.

Now, we continue our description by using the last result. Thus, in total, we obtain

$$\sum_{E\in\binom{[n]}{r}:n\in E} P(E) = \underbrace{q_n}_{=:q_n^1} + \underbrace{\sum_{1\leq i_1<n} q_{i_1,n}}_{=:q_n^2} + \underbrace{\sum_{\substack{1\leq i_1,i_2<n \\ i_1\neq i_2}} q_{i_1,i_2,n}}_{=:q_n^3} + \ldots + \underbrace{\sum_{(i_1,i_2,\ldots,i_{r-1})\in I_1^{r-1}} q_{i_1,i_2,\ldots,i_{r-1},n}}_{=:q_n^r}$$

with q_n^i denoting the probability of bin n to be drawn in the i-th sequential step of Algorithm 1 (recall that the algorithm samples an r-tuple from the set $[n]$). By the following lemma, we show the imbalance occurring with these probabilities.

Lemma 3.32. *For any step $t \geq 2$ of a given r-tuple, it holds that $q_n^t < c_n$.*

Proof. The proof follows by induction. For the basis, we consider $t = 2$. Then, according to Definition 3.8 of conditional probabilities, for arbitrary bins $i_1, i_2 \in [n]$, $i_1 \neq i_2$, the probability $\Pr[E_{i_1}^1 \cap E_{i_2}^2]$ to receive a copy each is $q_{i_1,i_2} = c_{i_2} \cdot \frac{c_{i_1}}{1-c_{i_1}}$. Thus,

$$q_n^2 \;=\; \sum_{i_1=1}^{n-1} q_{i_1,n} \;=\; c_n \cdot \sum_{i_1=1}^{n-1} \frac{c_{i_1}}{1-c_{i_1}}.$$

From this, it follows:

$$q_n^2 \;=\; c_n \cdot \sum_{i_1=1}^{n-1} \frac{c_{i_1}}{1-c_{i_1}} \overset{(c_1<c_n)}{<} c_n \cdot \sum_{i_1=1}^{n-1} \frac{c_{i_1}}{1-c_n} \;=\; c_n \cdot \frac{\sum_{i_1=1}^{n-1} c_{i_1}}{1-c_n} \;=\; c_n \cdot \frac{1-c_n}{1-c_n} \tag{3.10}$$

that is, $q_n^2 < c_n$ (analogues, one could show: $q_1^2 > c_1$).

Now, let $t > 2$. Then again, consider arbitrary bins $i_1, i_2, \ldots, i_t \in [n]$, $i_a \neq i_b \; \forall a \neq b \in [t]$, $a \neq b$, such that we calculate $\Pr[E_{i_1}^1 \cap E_{i_2}^2 \cap \ldots \cap E_{i_t}^t]$ as

$$q_{i_1,i_2,\ldots,i_t} = c_{i_t} \cdot \prod_{\ell=1}^{t-1} \frac{c_{i_\ell}}{1-\left(\sum_{m=1}^{\ell} c_{i_m}\right)}.$$

Since this term calculates the probability only for a fixed set of chosen bins, we continue with calculating the absolute probability q_n^t of bin n to be drawn in the t-th step which considers all possible combinations of bins to be drawn for allocation in the $t-1$ previous steps. For our description, we again refer to the index sets I_ℓ^u as defined in (3.9). Then, we obtain

$$\begin{aligned} q_n^t \;&=\; \sum_{(i_1,\ldots,i_{t-1},n)\in I_1^t} q_{i_1,\ldots,i_{t-1},n} \;=\; c_n \cdot \sum_{(i_1,\ldots,i_{t-1},n)\in I_1^t} \prod_{\ell=1}^{t-1} \frac{c_{i_\ell}}{1-\left(\sum_{m=1}^{\ell} c_{i_m}\right)} \\ &=\; c_n \cdot \sum_{(i_1,\ldots,i_{t-2},n)\in I_1^{t-1}} \sum_{\substack{1\leq i_{t-1}\leq n \\ i_{t-1}\neq i_1,\ldots,i_{t-2},n}} \prod_{\ell=1}^{t-1} \frac{c_{i_\ell}}{1-\left(\sum_{m=1}^{\ell} c_{i_m}\right)} \end{aligned}$$

We can now factor out the first $t-2$ factors because these factors and the indices $i_1, \ldots, i_{t-2}$ do not depend on the index i_{t-1}. For convenience, we shorten the description by defining

$$F(j_s) := \prod_{\ell=1}^{s} \frac{c_{i_\ell}}{1-\left(\sum_{m=1}^{\ell} c_{i_m}\right)}.$$

Then, we obtain

$$q_n^t \;=\; c_n \cdot \sum_{(i_1,\ldots,i_{t-2},n)\in I_1^{t-1}} F(i_{t-2}) \cdot \sum_{\substack{1\leq i_{t-1}\leq n \\ i_{t-1}\neq i_1,\ldots,i_{t-2},n}} \frac{c_{i_{t-1}}}{1-c_{i_1}-c_{i_2}-\ldots-c_{i_{t-1}}} \tag{3.11}$$

It now remains to find an appropriate estimate for the innermost sum because then, the calculation can be reduced to the previous step of the induction. Since only the innermost sum depends on index i_{t-1}, again, we substitute the capacity $c_{i_{t-1}}$ in the denominator of the innermost sum by c_n yielding the following inequality:

$$\begin{aligned}
q_n^t &= c_n \cdot \sum_{(i_1,\ldots,i_{t-2},n)\in I_1^{t-1}} F(i_{t-2}) \cdot \sum_{\substack{1\le i_{t-1}\le n \\ i_{t-1}\neq i_1,\ldots,i_{t-2},n}} \frac{c_{i_{t-1}}}{1-c_{i_1}-\ldots-c_{i_{t-2}}-c_{i_{t-1}}} \\
&\le c_n \cdot \sum_{(i_1,\ldots,i_{t-2},n)\in I_1^{t-1}} F(i_{t-2}) \cdot \sum_{\substack{1\le i_{t-1}\le n \\ i_{t-1}\neq i_1,\ldots,i_{t-2},n}} \frac{c_{i_{t-1}}}{1-c_{i_1}-\ldots-c_{i_{t-2}}-c_n} \\
&= c_n \cdot \sum_{(i_1,\ldots,i_{t-2},n)\in I_1^{t-1}} F(i_{t-2}) \cdot \frac{\sum_{\substack{1\le i_{t-1}\le n \\ i_{t-1}\neq i_1,\ldots,i_{t-2},n}} c_{i_{t-1}}}{1-c_{i_1}-\ldots-c_{i_{t-2}}-c_n} \\
&= c_n \cdot \sum_{(i_1,\ldots,i_{t-2},n)\in I_1^{t-1}} F(i_{t-2}) \cdot \frac{1-c_n-\left(\sum_{l=1}^{t-2} c_{i_l}\right)}{1-c_n-\left(\sum_{l=1}^{t-2} c_{i_l}\right)}
\end{aligned}$$

The reduction continues until reaching the induction base that contains a $<$-relation as shown in Inequality 3.10, thus, we obtain $q_n^t < c_n$ and the lemma follows. ■

Lemma 3.32 shows that for any step $t \ge 2$ of allocation, $q_n^t < c_n$ yielding the following result:

$$\sum_{E\in\binom{[n]}{r}:n\in E} P(E) \;=\; q_n^1 + q_n^2 + \ldots + q_n^r \;<\; r\cdot c_n \;=\; p_n$$

satisfying Inequality 3.7 and thus, the theorem follows. ■

3.3.2.3 Determining Upper Bounds for the Imbalance

Since by Lemma 3.32, we know about the existence of the imbalance for any step $2 \le t \le r$ if Algorithm 1 is used for allocation on non-uniform capacities, in this section, we are furthermore interested in determining the size of the imbalance, i.e. its maximum deviation from an optimal allocation. More precisely, to obtain an even data layout, Algorithm 1 usually supposes any bin i to be drawn with allocation probability c_i in every allocation step t. Unfortunately, as we have shown, the desired data layout can only be achieved if solely uniform capacities are considered, otherwise, we obtain a biased data distribution. In other words, in a uniform setting, the imbalance equals zero, whereas with non-uniform capacities, there is a deviation from a fair layout (Theorem 3.30). Moreover, the more the capacities of the bins deviate from being uniform the greater the imbalance. Therefore, obviously, the imbalance directly depends on the underlying disk configuration.

As the imbalance in the system is determined for each disk exclusively, we now calculate the maximum deviation with respect to bin n that we assume to have maximum capacity $c_n = \frac{1}{r}$. Clearly, this could be of practical interest for e.g. administrators who always aim at providing the best access performance of the administrated storage network, even if the SAN consists of a mixture of heterogeneous capacities, like e.g. some 80 to 500 GByte disks combined with a number of 3 TByte RAID-systems. Then, condition to a required total storage capacity and with knowledge about the sizes of the particularly induced imbalances, the administrator can decide what combination of storage devices to chose in order to ease the imbalance and thus, to increase system performance.

Since the deviation is determined according to a fixed capacity distribution for which exact values are needed, for our calculations we first define some capacity range for the disk capacities. Then, given this range, we can calculate the induced imbalance (for this, recall Lemma 3.29 for denoting the maximum size of a disk).

Definition 3.33. *Consider a storage network $S(n, C[n])$, a share tuple $\bar{c}$ and a replication parameter $r \geq 2$. Then, a capacity distribution $c_1, \ldots, c_n$ is called* ***γ-bounded*** *if any capacity c_i in $\bar{c}$ is taken from the range $\frac{1}{\gamma r} \leq c_i \leq \frac{1}{r}$ for an arbitrary $\gamma \in \mathbb{Q}_{>1}$.*

Due the proof of Lemma 3.32, the task now is to determine a value for the $<$-relation in each step t, as first shown in Inequality 3.10. Again, as different steps induce different degrees of index dependencies, for different steps $2 \leq t, t' \leq r$, $t \neq t'$, we have to consider different functions for our calculations. Obviously, since regarding a $<$-relation, the imbalance occurring in any step $t \geq 2$ becomes worst if the corresponding function reaches its global minimum. Furthermore, since first, in a uniform setting, there is no imbalance, and second, all bins $i \in [n]$ are supposed to have capacity $c_i \leq \frac{1}{r}$, we start our considerations from the smallest setting of bins which is given by a set of r bins of capacity $\frac{1}{r}$ each. Then, our analysis is as follows.

Again, we fix bin n of capacity $c_n = \frac{1}{r}$ which, initially, holds for also for all other bins and that we assume to remain unchanged for the whole process. Then, for each step $t \geq 2$, the task is to find the minimum over all possible γ-bounded capacity distributions on the remaining bins. Clearly, as we fixed bin n, the total capacity that is provided by any disk configuration consisting of $r-1$ additional disks is always $1 - \frac{1}{r}$. Moreover, since the initial configuration is the only one to provide this total capacity by solely r bins, any other capacity distribution force them to grow by number. We start with the initial configuration consisting of bin n and $r-1$ additional bins, each having capacity $\frac{1}{r}$. Then, we determine the global minimum of any possible capacity distribution on these $r-1$ remaining bins. After that, we proceed with calculating the global minimum for any capacity distribution on r bins (with 2 bins of size $< \frac{1}{r}$), and so forth, until finally, we reach the maximum number of remaining bins, denoted by $n-1$ in the following, in which all bins have capacity $\frac{1}{\gamma r}$.

The Case $t = 2$

We start with stating the size of the imbalance for step $t = 2$. That is, we consider the calculation for the probability q_n^2 as given in Lemma 3.32 for which we determine the minimum over all γ-bounded capacity distributions on the set of remaining bins in the system (recall that to ensuring the total capacity $1 - \frac{1}{r}$ by different distributions a variable number of bins in different steps must be considered). More formally, we calculate

$$\min_{r-1 \leq \ell \leq n-1} \left\{ \min_{c_1,\ldots,c_\ell} \sum_{i_1=1}^{\ell} \frac{c_{i_1}}{1-c_{i_1}} \right\}. \tag{3.12}$$

Then, the following theorem shows that the imbalance becomes worst if the configuration consists of n bins, that is, for $n-1$ bins the capacity is $\frac{1}{\gamma r}$.

Theorem 3.34. *For all $i \in [n]$, let $c_i \in [\frac{1}{\gamma r}, \frac{1}{r}]$, and fix $c_n = \frac{1}{r}$. Then, Equation 3.12 is minimal if $\ell = n-1$ and $c_i = \frac{1}{\gamma r}$ $\forall i \in [n-1]$.*

Proof. Due to step $t = 2$, we denote the innermost sum by $X^{(2)} := \sum_{i_1=1}^{\ell} \frac{c_{i_1}}{1-c_{i_1}}$. Furthermore, we consider the capacity $c^* := (1 - \frac{1}{r})/\ell$ which defines an even distribution of the total capacity $1 - \frac{1}{r}$ covered by the ℓ observed bins. It is easy to show that the innermost term $\frac{c_{i_1}}{1-c_{i_1}}$ corresponds to a convex and strictly monotonically increasing function on the compact interval $[\frac{1}{\gamma r}, \frac{1}{r}]$ implying that $X^{(2)}$ is also convex and strictly monotonically increasing for each ℓ, and moreover, this means that Equation 3.12 has a unique solution. With this, we can state our assumption by using the following lemma to show that $X^{(2)}$ is minimal if and only if all bins have size c^*.

Lemma 3.35. *$X^{(2)}$ is minimal if and only if for all $i, j \in [\ell]$, $i \neq j$, it holds that $c_i = c_j$.*

Proof. The proof is straightforward. We assume that at least two indices i, j, $i \neq j$, exist such that $c_i = c^* + \varepsilon$ and $c_j = c^* - \varepsilon$ for an arbitrary $\varepsilon \geq 0$. Then, we simply have to find the minimal ε for

$$\frac{c_i}{1-c_i} + \frac{c_j}{1-c_j} \geq \frac{2c^*}{1-c^*}.$$

For this, we define the following function due to c_i and c_j

$$f(\varepsilon) := \frac{c^* + \varepsilon}{1 - c^* - \varepsilon} + \frac{c^* - \varepsilon}{1 - c^* + \varepsilon},$$

and to find the minimum of $f(\varepsilon)$, we have to look at the first and second order derivatives $f'(\varepsilon)$ and $f''(\varepsilon)$, respectively. With

$$f'(\varepsilon) = \frac{1}{(1 - c^* - \varepsilon)^2} - \frac{1}{(1 - c^* + \varepsilon)^2}$$

we obtain $f'(\varepsilon) = 0$ if and only if $\varepsilon = 0$. Furthermore, for $\varepsilon = 0$, $f''(\varepsilon) > 0$, and hence, the lemma follows. ■

The result from Lemma 3.35 implies that the minimum is obtained if $X^{(2)} = \sum_{i_1=1}^{\ell} \frac{c^*}{1-c^*}$ which can then be transformed into the following term that we denote by Y:

$$Y := \frac{1 - \frac{1}{\ell}}{1 - \left(1 - \frac{(1/r)}{\ell}\right)}$$

Since Y is monotonically decreasing in ℓ, its minimum is given if $\ell = n-1$ and hence, the theorem follows. ■

So far, we showed that for step $t = 2$ the imbalance on the data distribution becomes worst if there is at most one bin of maximum size $\frac{1}{r}$ and all remaining bins have smallest size $\frac{1}{\gamma r}$. Thus, it remains to show the same for the subsequent steps.

The Case $t > 2$

According to the proof of Theorem 3.30, to calculate the allocation bias for any subsequent step t, $3 \leq t \leq r$, we again consider the functions for the probability q_n^t as follows (for better description, we again use the sets I_1^t, as defined in (3.9)).

$$\min_{r-1 \leq \ell \leq n-1} \left\{ \min_{c_1,\ldots,c_\ell} \underbrace{\sum_{(i_1,\ldots,i_{t-1},n) \in I_1^t} \left(\prod_{\ell=1}^{t-1} \frac{c_{i_\ell}}{1 - \left(\sum_{m=1}^{\ell} c_{i_m}\right)} \right)}_{=:X^{(t)}} \right\} \tag{3.13}$$

For this problem and any observed function, we again suppose the global minimum if $\ell = n-1$, that is, if $n-1$ bins have minimum capacity $\frac{1}{\gamma r}$ but this is only true if the following conjecture holds (for which no proof is given so far).

Conjecture 3.36. *For any step $t \geq 3$, $X^{(t)}$ is a convex and strictly monotonically increasing function on the compact interval $[\frac{1}{\gamma r}, \frac{1}{r}]$.*

Corollary 3.37. *If and only if Conjecture 3.36 is true, the minimum for Equation 3.13 is given if $\ell = n-1$ and $c_i = \frac{1}{\gamma r}$ $\forall i \in [n-1]$.*

Theorem 3.38. *Consider a storage network $S(n, C[n])$ with γ-bounded capacities for some $\gamma \in \mathbb{Q}_{>1}$, a replication parameter r and a fixed capacity $c_n = \frac{1}{r}$. If Corollary 3.37 holds, then the maximum allocation imbalance for bin n in any step $2 \leq t \leq r$ is given by*

$$q_n^t = \frac{1}{r} \cdot \prod_{i=1}^{t-1} \frac{n-i}{\gamma r - i} \quad \text{if } \gamma \in \mathbb{Q}_{>1},$$

and

$$q_n^t = \frac{1}{r} \cdot \left[\binom{n-1}{t-1} \Big/ \binom{\gamma r - 1}{t-1} \right] \quad \text{if } (\gamma r) \in \mathbb{Z}_{>1}.$$

Proof. The proof is given by induction on t. Particularly, we show for each t that $q_n^t < \frac{1}{r}$, and further, that for any step $t > 2$, $q_n^t = \frac{1}{r} \cdot \prod_{i=1}^{t-1} \frac{n-i}{\gamma r - i} < q_n^{t-1}$.

We start with $t = 2$. Furthermore, we assume n disks with $c_n = \frac{1}{r}$ and $c_j = \frac{1}{\gamma r}$ for all $j \in [n-1]$. Then, by replacing the values in Equation 3.10 (in proof of Lemma 3.32), the first assumption holds since

$$q_n^2 = \frac{1}{r} \cdot \sum_{i_1=1}^{n-1} \frac{\frac{1}{\gamma r}}{1 - \frac{1}{\gamma r}} = \frac{1}{r} \cdot \frac{n-1}{\gamma r - 1} \overset{(\gamma r > n)}{<} \frac{1}{r}$$

which holds because as n is fixed, $\frac{1}{\gamma r} < \frac{1}{n}$ in the non-uniform case. To show this, recall that $\frac{1}{\gamma r}$ is the smallest capacity to be chosen for any bin. Then, condition to a fixed n, a total capacity of at least $n \cdot \frac{1}{\gamma r} \leq 1$ is provided. If we now assume $\frac{1}{\gamma r} \geq \frac{1}{n}$, it directly follows that $\frac{1}{\gamma r} = \frac{1}{n}$ implying the uniform case what is a contradiction.

Now, consider $t > 2$ and assume for all previous steps $t - j$, $j \geq 1$, that $q_n^{t-j} < \frac{1}{r}$ holds. Then, according to Equation 3.11 (in Lemma 3.32), we consider

$$q_n^t = c_n \cdot \sum_{(i_1,\ldots,j_{t-2},n) \in I_1^{t-1}} F(i_{t-2}) \cdot \sum_{\substack{1 \leq i_{t-1} \leq n \\ i_{t-1} \neq i_1,\ldots,i_{t-2},n}} \frac{c_{i_{t-1}}}{1 - c_{i_1} - c_{i_2} - \ldots - c_{i_{t-1}}}$$

and as we suppose $c_i = \frac{1}{\gamma r}$ for all $i \neq n$, we can substitute as follows:

$$\begin{aligned} q_n^t &= c_n \cdot \sum_{(i_1,\ldots,j_{t-2},n) \in I_1^{t-1}} F(i_{t-2}) \cdot \sum_{j=1}^{n-t+1} \frac{\frac{1}{\gamma r}}{1 - \underbrace{\frac{1}{\gamma r} - \frac{1}{\gamma r} - \ldots - \frac{1}{\gamma r}}_{t-1 \text{ elements}}} \\ &= c_n \cdot \sum_{(i_1,\ldots,j_{t-2},n) \in I_1^{t-1}} F(i_{t-2}) \cdot \frac{n-t+1}{\gamma r - t + 1} \end{aligned}$$

As $q_n^{t-1} = c_n \cdot \sum_{(i_1,\ldots,j_{t-2},n) \in I_1^{t-1}} F(i_{t-2}) < \frac{1}{r}$ by assumption, the induction follows. Moreover, again, since $\gamma r > n$, it follows that $\frac{n-t+1}{\gamma r - t + 1} < 1$, and thus, $q_n^t < q_n^{t-1}$.

If $\gamma r \in \mathbb{Z}_{>1}$, the second result can simply be calculated as

$$q_n^t = \frac{1}{r} \cdot \frac{n-1}{\gamma r - 1} \cdot \frac{n-2}{\gamma r - 2} \cdot \ldots \cdot \frac{n-(t-1)}{\gamma r - (t-1)} = \frac{1}{r} \cdot \frac{\binom{n-1}{t-1}}{\binom{\gamma r - 1}{t-1}}$$

■

So far, we showed that if sampling some edge $E \in \binom{[n]}{r}$ according to the distribution P by using a common balls-into-bins scheme (Algorithm 1), a redundant data layout combined with an optimal utilization of the offered system capacity can never be achieved for heterogeneous environments. However, in the following sections, we show that the desired data distribution can be obtained by placing the data blocks somewhat more controlled, and what holds for static as well as dynamic storage environments.

3.4 COMB: An Allocation Strategy for Static Scenarios

Although we showed in the previous section that applying a standard balls-into-bins approach for redundant data allocating can achieve fairness only for homogeneous environments, there are numerous results making such approaches very suitable for random allocation because, as we briefly sketched in Section 3.3.2.1, the results are quite convincing to linger on them. Thus, we now show that introducing some slight modifications to the data placement in Algorithm 1 suffices to obtain the desired data layout.

Before we start our description, we first briefly discuss some intuitive approaches for introducing redundancy into the standard balls-into-bins scheme.

Naive Approaches.

1. Allocate all balls independently by some common balls-into-bins scheme but ignore all trials in which the same bin is drawn for identical copies such that, finally, all elements of some certain edge $E \in \binom{[n]}{r}$ are stored separately.

2. Since, according to Definition 3.5, fairness requires load $\ell_i = r \cdot k \cdot c_i$ for all bins $i \in [n]$, one could try to partition the set $[n]$ into r disjoint subsets of size $\frac{1}{r}$ each and then allocate the elements of each edge $E \in \binom{[n]}{r}$ such that each partition gets at most one copy.

We call these approaches naive because it is quite obvious that they do not provide the desired result. In particular, if we consider non-uniform disk capacities for the first approach, then, after allocating all balls to the disks, the valid trials (i.e. the ones that were not ignored) again lead to a biased data placement according to the result of Theorem 3.30. The second scheme also runs into problems because the mentioned approach corresponds to the well-known PARTITION problem which is NP-complete.

Therefore, in this section, we introduce an efficient placement scheme that yields the desired layout by just sampling a hypergraph edge E according to a given probability distribution P. Particularly, for a storage network $S(n, C[n])$ and the following allocation strategy, P is simply defined by the values of the corresponding share tuple $\bar{c}$. With respect to Lemma 3.26 and Theorem 3.27, this is feasible because we sample r-tuples and as we define $p_i = r \cdot c_i \in [0,1]$ for each bin i and $\sum_{i=1}^{n} c_i = 1$, the necessary conditions to obtain a fair distribution are satisfied (surely, provided that $c_i \leq \frac{1}{r}$).

The following algorithm is called *COMB strategy*, and it is closely related to a redundant balls-into-bins model but differs due to the fact that only the first of r identical copies is always allocated at random to the set $[n]$ of bins, and all subsequent ones are placed deterministically on distinct bins. Moreover, we show first that the resulting load of any bin corresponds to a standard single-choice balls-into-bins allocation, and second, that it is highly concentrated around the expected value.

In particular, consider a storage network $S(n, C[n])$ and a corresponding share tuple $\bar{c}$ for the disks in the set $[n]$. Let the bins be numbered in a range from 1 to n and arranged in a linear manner in the $[0,1)$-interval (that we consider as a ring) according to their shares in $\bar{c}$. That is, bin i is responsible for the half-open interval $I_i = [\alpha_i, \alpha_i + c_i) \subseteq [0,1)$ of length c_i starting at point $\alpha_i = \sum_{j=1}^{i-1} c_j$. Now, given a position $x \in [0,1)$, let $I_C(x) \in [n]$ denote the index of the interval I_i which x lies in. Then, the COMB strategy works as follows.

Algorithm 2 The COMB Strategy

Require: Probability space $(S(n, C[n]), \bar{c})$, Set $[k]$ of balls, Replication parameter r
Ensure: A set $M_k^r = \{(I_1^1, I_2^1, \ldots, I_r^1), \ldots, (I_1^k, I_2^k, \ldots, I_r^k)\}$ of r-tuples of indices of bins.

1: **for all** $d \in [k]$ **do**
2: Place d uniformly at random at position $x_1 \in [0,1)$ yielding interval $I_1 = I_C(x_1)$
3: Set $i \leftarrow 2$
4: **while** $i \leq r$ **do**
5: Allocate d to position $x_i = (x_1 + (i-1) \cdot \frac{1}{r})$ yielding interval $I_i = I_C(x_i)$
6: Set $i \leftarrow (i+1)$

The COMB strategy, which works on disk assignments into the $[0,1)$-interval, is graphically illustrated in Figure 3.2 for a single 4-tuple. The first copy is allocated uniformly at random to some point $x_1 \in [0,1)$, and all subsequent copies $i \in \{2,3,4\}$ are deterministically placed to locations x_i by a distance of $(i-1) \cdot \frac{1}{4}$ from x_1 in the unit ring. In the following theorem, we show the correctness and layout quality of the COMB algorithm.

Theorem 3.39. *For a static storage network $S(n, C[n])$ of arbitrary capacities, a set $[k]$ of distinct data blocks and $r \geq 2$, the COMB strategy provides a fair and redundant distribution of $m = k \cdot r$ replicated balls on the set $[n]$ of bins. Moreover, any bin i gets at most*

$$\ell_i = c_i \cdot k \cdot r + O(\sqrt{c_i \cdot k \cdot r \cdot \log n})$$

data blocks w.h.p. if $\sqrt{3 \cdot \ell \cdot \frac{\log n}{c_i \cdot k \cdot r}} < 1$ for some constant $\ell > 0$.

Proof. We prove the correctness of the COMB strategy in terms of redundancy and a balanced data load.

Redundancy. As any bin $i \in [n]$ is assigned a half-open interval I_i of size $c_i \leq \frac{1}{r}$, and furthermore, all copies of any data block $d \in [k]$ are allocated by a distance of $\frac{1}{r}$ from each other[7], no half-open interval can simultaneously contain two different locations x_j, x_{j+1} for any two identical copies. Thus, all copies d_j, $j \in [r]$, lie in different intervals, and from this it follows that all values $j_1, j_2, \ldots, j_r$ in the sampled r-tuple are pairwise distinct. If

[7] More precisely, any distance in the interval $[\max_i\{I_i\}, \frac{1}{r}]$ is valid to achieve the desired data layout.

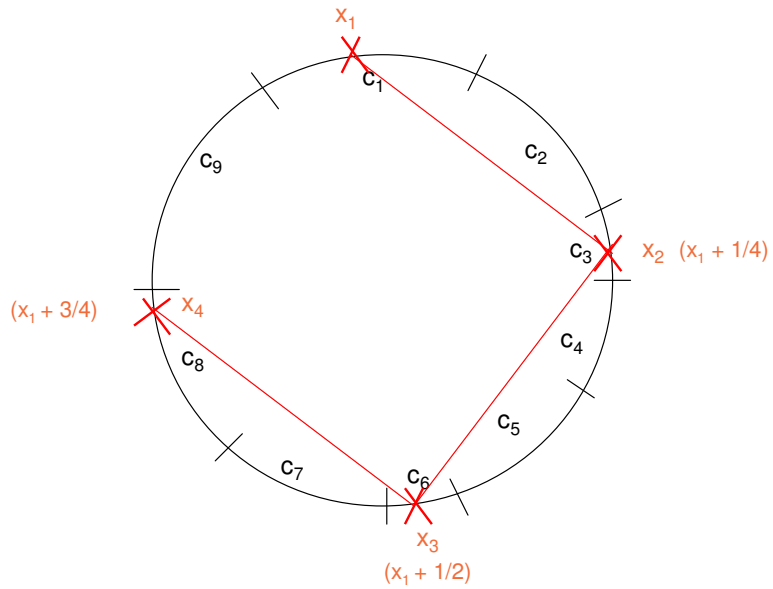

Figure 3.2: The COMB strategy on the $[0,1)$-interval for $n = 9$ bins and $r = 4$.

then some function $f : [n]^r \to \binom{[n]}{r}$ is applied that maps each sampled r-tuple to some edge $E \in \binom{[n]}{r}$, this yields the required r-element set.

Fairness. Given the set $[k]$ and parameter r, we obtain k distinct r-tuples independently sampled by the COMB strategy. As, for each such tuple, the first location x_1 is chosen uniformly at random in the $[0,1)$-ring, clearly, the probability for some interval I_i to be chosen is $\Pr[I_C(x_1) = I_i] = c_i$. Moreover, if x_1 is chosen uniformly random, then so is $x_j = x_1 + \frac{j-1}{r}$ for each $j \in \{2,\ldots,r\}$; therefore, regardless of the obvious correlation between x_1 and x_j, and depending on the size c_ℓ for some interval I_ℓ to be chosen, it holds that $\Pr[I_C(x_j) = I_\ell)] = c_\ell$ for all $\ell \in [n]$.

For the rest of this proof, consider an arbitrary but fixed bin i with allocation probability c_i. Furthermore, for an arbitrary r-tuple S, $1 \leq S \leq k$, let X_i^S be the binary random variable denoting the event that bin i is drawn by S, and let $X_i = \sum_S X_i^S$. Then, referring to Question 3.25, it holds that $\Pr[X_i^S = 1] = r \cdot c_i\,(= p_i)$ which satisfies the necessary condition in Lemma 3.26 for balanced allocations. Additionally, since each random variable X_i is binomially distributed, its expected value $\mathrm{E}[X_i] = r \cdot k \cdot c_i$ which, on expectation, ensures a fair data layout according to Definition 3.5.

At last, it remains to show that any disk contains the expected number of blocks also w.h.p. Since all r-tuples are mapped to the $[0,1)$-interval independently at random and the X_i^S's are binary random variables, we can apply the Chernoff bounds from Lemma 3.15 gaining the following upper bound on the load of bin i:

$$\Pr[X_i \geq (1+\varepsilon) \cdot c_i \cdot k \cdot r] \leq e^{-min[\delta^2,\varepsilon] \cdot c_i \cdot k \cdot r/3} \tag{3.14}$$

for every $\varepsilon > 0$.

To prove the bound on the concentration, we need to derive the particular ε for which the probability drops below $n^{-\ell}$ (we exemplarily sketch the calculation for obtaining the correct ε once in this thesis). Considering Equation 3.14, from the right hand side, we get the following, assuming $\varepsilon < 1$,

$$\begin{aligned} e^{-\varepsilon^2 \cdot c_i \cdot k \cdot r/3} &\leq n^{-\ell} \\ n^{\ell} &\leq e^{\varepsilon^2 \cdot c_i \cdot k \cdot r/3} \end{aligned}$$

Setting $\varepsilon = \sqrt{\varepsilon' \cdot \log n}$ leads to

$$\begin{aligned} n^{\ell} &\leq e^{\left(\sqrt{\varepsilon' \cdot \log n}\right)^2 \cdot c_i \cdot k \cdot r/3} \\ n^{\ell} &\leq e^{\log n \cdot \varepsilon' \cdot c_i \cdot k \cdot r/3} \\ n^{\ell} &\leq n^{\varepsilon' \cdot c_i \cdot k \cdot r/3} \\ \varepsilon' &\geq \frac{3 \cdot \ell}{c_i \cdot k \cdot r} \end{aligned}$$

Substituting this in Inequality 3.14 concludes the proof. ■

The above result shows that for bin i with capacity c_i the maximum deviation of the load from its expected value is in $O(\sqrt{c_i \cdot k \cdot r \cdot \log n})$. Recall that this directly corresponds to the deviation of a standard single-choice balls-into-bins game (Section 3.3.2.1). Thus, as the COMB strategy implements some sort of redundant single-choice balls-into-bins approach, the achieved result is the best to obtain for a redundant single-choice process. However, like in the standard (non-redundant) case, the deviation heavily depends on the total number of balls allocated to the bins, that is, the more balls thrown the worse the deviation. However, we leave a possible improvement of this drawback as the following

Open Problem: *Is it possible to improve the presented single-choice deviation bound by some multiple-choice allocation algorithm that allocates r identical copies to one out of $d \geq 2$ sampled r-tuples (that are determined by their first copy allocations similar to the shown COMB strategy) ?*

Furthermore, as a more essential drawback, the COMB strategy neither is nor can be made adaptive. As we have shown, the allocation process works on an n-fold partitioning of the $[0,1)$-interval into n intervals of lengths $c_1, c_2, \ldots, c_n$, respectively. Since $\sum_i c_i = 1$, it is easy to see that this scheme works efficiently for static environments only because if the number of disks changes, the system obviously runs into fragmentation problems. That is, whenever a new disk enters the system, this would affect the sizes of all intervals (even if the disk has only small capacity) what in turn leads to a huge amount of re-allocations and thus, to intolerable performance deficits over a certain time.

As a consequence, we address the problem of occurring dynamics in the storage network in more detail in the following section.

3.5 SPREAD: An Adaptive Allocation Scheme for Dynamic Scenarios

We introduce an advanced data placement scheme called *SPREAD* [MS08] which, unlike the former strategies, provides useful adaptivity properties to face dynamic changes in scalable SANs, that is, it is capable of handling any occurring changes in the capacity distribution on the disks in the storage network that may be caused by faulty disks or growing storage demands. More precisely, we present a redundant data placement algorithm that

- satisfies redundancy, fairness and adaptivity in a time- and space-efficient manner,
- is able to locate any stored data item efficiently (access time is at most logarithmic in the system size),
- uses lookup functions whose space requirements only depend on the number of disks in the system and not on the number of data items or capacity distribution,
- given a replication parameter r, may cope with a variable number r_d of copies per data item d as long as $r_d \leq r$.

In contrast to the algorithms described in the last two sections that primarily focused on the obtained data layout without giving a practical notion of the allocation functions, we now consider the practical realization of such functions as well as used data structures and lookup-times for realizing the dictionary functions *search(T,k)* and *insert(T,k)* (c.f. Section 3.1.2.1).

As we know, the most efficient approach to implement the dictionary functions is provided by *hashing*, where some given set $P \subseteq U$ of data blocks with keys taken from a universe U of all possible keys is distributed sufficiently even over a set of disks in a storage network by applying an appropriate hash function h (see Figure 3.3). More particularly, for achieving an appropriate even allocation of data items to the disks, all strategies we present in this section apply k-universal hash functions (recall from Section 3.1.2.2 that $k = O(\log n)$ suffices for our demands). Again, the very advantage we gain by this way of data placement is that allocations can be done with complexity $O(1)$. However, as hashing is usually utilized for static environments, and as we have further shown with Linear Hashing in Section 2, a simple substitution of the hash functions in case of a scaling environment is not sufficient for satisfying our demands. Thus, we are left with the following question: *How to design an adaptive hash function ?*

Before we describe the SPREAD strategy in more detail, we first tackle this design problem by briefly introducing the most important adaptive strategies that have been published so far and discuss why it is so difficult to adapt these so that all conditions are met. Furthermore, note that the adaptive schemes we introduce in the following differ from

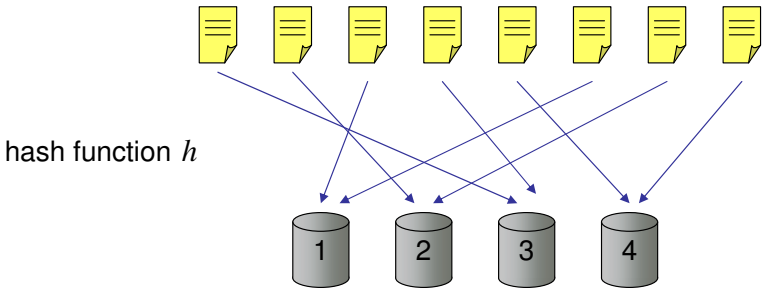

Figure 3.3: Hashing

SPREAD with respect to the fact that none of them is designed to handle redundancy either appropriately or at all, that is, they feature no or only less efficient fault-tolerant mechanisms. Thus, we simply ignore the redundancy condition for a moment.

3.5.1 Previous Adaptive Strategies

If all storage devices have the same capacity, surely, we are not concerned with the heterogeneity issue such that any strategy has to assign each disk simply the same data load. Thus, in a static environment, a standard balls-into-bins algorithm could easily provide the desired fair distribution (and COMB if redundancy is demanded), but as we consider dynamic scenarios, we are left with the difficulty of changes in the environment, that is, the addition and removal of disks, respectively data items. Nevertheless, the usage of only uniform capacities simplifies the allocation problem considerably.

Exemplary, we restrict our descriptions on two prominent schemes, called *Consistent Hashing* and *SHARE*, which were the first approaches introduced to feature adaptivity for uniform and also non-uniform capacity distributions.

3.5.1.1 Consistent Hashing for Uniform Capacities

Consistent Hashing, which is also called *Nearest Neighbor Strategy*, was published by Karger et al.[KLL$^+$97]. Initially, the scheme was part of a global distribution scheme for web-based contents, and it has further been applied to Web caching in [KSB$^+$99].

In its basic form, Consistent Hashing uses two hash functions, $g : U \to [0,1)$ for mapping the (pairwise independent) data items into the $[0,1)$-interval (which again is considered as a modulo 1 ring) and $h : V \to [0,1)$ to map the nodes (resp. storage devices) into the same $[0,1)$-ring. Then, each node v is responsible for the interval $[h((v), h((w)) \subseteq [0,1)$, with some node w being the direct successor of v in the mapping into the unit ring. Now, given the two functions, every data item $d \in U$ is placed to position $g(d)$ and furthermore stored at the node v with $h(v)$ being closest from above to $g(d)$, that is, it is assigned to the interval $[h((v), h((w))$ (in Figure 3.4, F_2 marks such interval for $v = 2$).

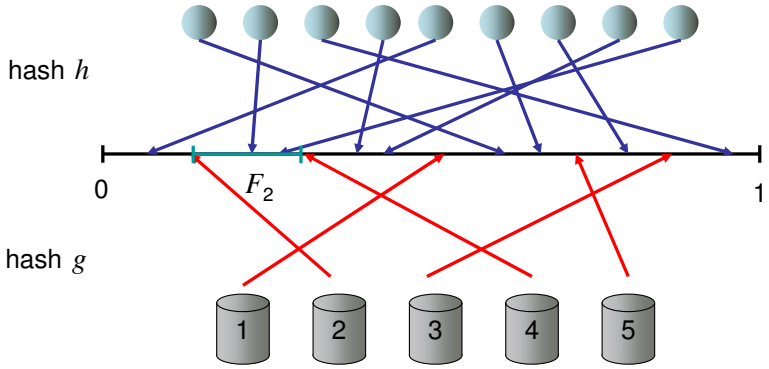

Figure 3.4: ConsistentHashing

In the basic case, if only one single function g is used for placing the nodes independently and uniformly into the $[0,1)$-ring, this would lead to a biased data layout as stated by the following lemma (the proof can be found in [MNR02]).

Lemma 3.40. *If n nodes are allocated to the $[0,1)$-ring independently and uniform at random, the length of the longest segment is w.h.p. $\Theta\left(\frac{\log n}{n}\right)$, and w.h.p. there is no segment which is shorter than $\Theta\left(\frac{1}{n^2}\right)$.*

Fortunately, this problem can be circumvented by using $O(\log n)$ differently labeled copies for each node v, all of which are allocated to the $[0,1)$-interval. If then g and h are $O(\log n)$-universal hash functions, it is easy to show that, first, Consistent Hashing yields an even distribution on the nodes, and second, the strategy is 2-competitive (see e.g. [KLL$^+$97, Sal04]). Moreover, also multiple copies for each data item d can easily be stored in a redundant way while preserving fairness by the rule that r copies of d are stored in the r nodes v with $h(v)$ being among the r closest successors of $g(d)$ in $[0,1)$. However, Consistent Hashing is no longer fair if applied to nodes of non-uniform capacities. This can be repaired by cutting the nodes into pieces of uniform capacity, but then the space for the hash table would depend on the differences in capacity and not just the number of nodes, which causes maintenance problems. In particular, it would be difficult to adapt this approach to changes in the minimum node capacity without inducing high data replacements and an undesired storage overhead, respectively.

Another adaptive hash table method for uniform storage devices, called *Cut-and-Paste Strategy*, has been presented in [BSS00, San01] for which it can be shown that, in contrast to Consistent Hashing, it keeps the deviation from the desired number of balls in a bin extremely small w.h.p. However, like Consistent Hashing, the Cut-and-Paste strategy is also hard to extend to the non-uniform case.

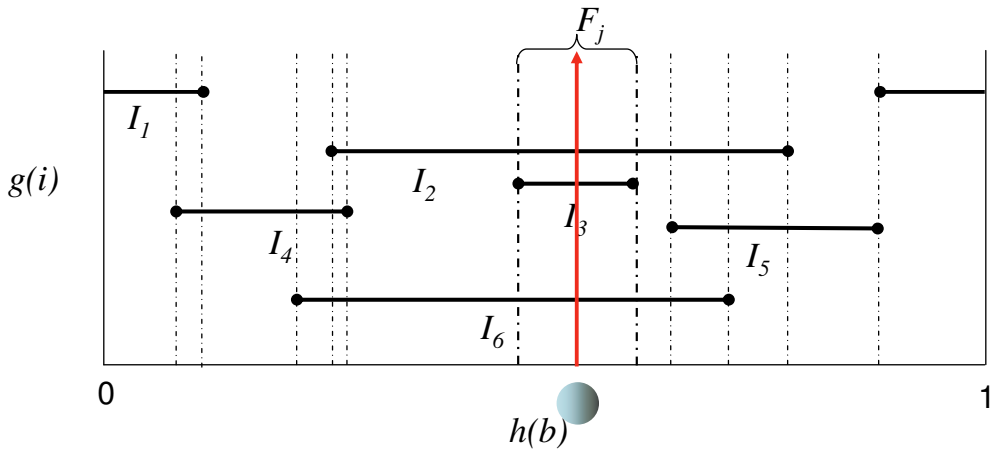

Figure 3.5: The SHARE Strategy

3.5.1.2 The SHARE Strategy for Non-Uniform Capacities

One of the first adaptive hash tables that can handle storage devices of arbitrary capacities was introduced in [BSS02], called *SHARE*, and which works in two phases. At first, there is a reduction phase implementing the conversion of the given non-uniform allocation problem into several uniform ones. In the second phase, a uniform allocation strategy, like e.g. Consistent Hashing, is then applied to each of these uniform problems to finally assign the data items to the bins.

In more detail, given any data item $b \in U$ for allocation, like in Consistent Hashing, the reduction phase of SHARE also applies two random hash functions, $h : U \to [0,1)$ to map the data items, and $g : V \to [0,1)$ to map the nodes into the $[0,1)$-interval (which again is considered as a modulo ring). Surely, both functions must ensure a sufficient even distribution of the items which, as we know, can be obtained by using $O(\log n)$-universal hash functions for h and g. Then, unlike in Consistent Hashing, each node $v \in V$ is assigned a subinterval I_v in the $[0,1)$-ring whose starting point is $g(v) \in [0,1)$ and that has length $s \cdot c_v$, where c_v denotes the relative capacity of the node v, and $s = \Theta(\log n)$ is a *stretch factor* that is needed to ensure a complete coverage of the $[0,1)$-interval by the subintervals, what, furthermore, guarantees high probability on the allocation. That is, any interval I_i that starts at some point $g(i)$ ends at point $(g(i) + s \cdot c_i) \mod 1$.

Furthermore, as depicted in Figure 3.5, given the hashed node intervals, each start- or endpoint of an interval marks the beginning or ending of some frame F_j. Since all such frames are disjoint in the ring, each data item b that is hashed to some location $h(b)$ is thus exclusively assigned to exactly one frame. Moreover, as the borders of the frames are defined by the start- or endpoints of the subintervals, each frame is completely covered by a number of subintervals. At last, the chosen uniform placement strategy is applied to any frame to select the target node out of the set of nodes assigned to each frame for

data placement. Then, the SHARE scheme yields the desired fair distribution on an either uniform or non-uniform as well as scalable storage environment w.h.p. and is $(2+\varepsilon)$-competitive, where ε can be made arbitrarily small.

Another approach that can also be found in [BSS02] and that is called *SIEVE* is organized as a multistage filter and overcomes some of the drawbacks inherent to SHARE, especially, it does not rely on an extra placement strategy for uniform capacities. However, to provide a fair placement, the SIEVE algorithm maintains a logarithmic number of random hash functions, one for each stage, and surely, keeping all these functions in memory leads to an undesired space consumption.

Nevertheless, for any capacity distribution, SHARE and SIEVE are efficient, $(2+\varepsilon)$-adaptive and both yield a fair data dispersal within a $(1\pm\varepsilon)$ factor, where $\varepsilon > 0$ can be made arbitrarily small. Particularly, the bound on the adaptivity of SHARE and SIEVE is based upon the following Fact 3.41. In more detail, for a hash table to be adaptive, it has to be able to handle any capacity change from $(c_1,\ldots,c_n)$ to $(c'_1,\ldots,c'_n)$, and as we already mentioned in Section 3.1.1, any adaptive storage strategy that wants to preserve fairness has to replace at least a

$$\sum_{i:c_i>c'_i}(c_i-c'_i)=\frac{1}{2}\sum_i|c_i-c'_i|$$

fraction of the data in the system, hence, the following fact holds.

Fact 3.41. *If, for any change from $(c_1,\ldots,c_n)$ to $(c'_1,\ldots,c'_n)$, a storage strategy only needs to replace a $d\sum_i|c_i-c'_i|$-fraction of the data in the system, then it is $2d$-adaptive w.r.t. capacity changes.*

Regarding fault-tolerance, obviously, the attempt to make SIEVE redundant is a hard challenge (if possible at all) but introducing redundancy into SHARE in some rudimentary way can be done as follows. If for some given stretch factor $s=\Theta(\log n)$ the capacity of every disk $i\in[n]$ is bounded by $c_i\leq\frac{1}{r\cdot s}$, then one can assign the copies of any data item d to positions $h(d), h(d)+\frac{1}{r}, h(d)+\frac{2}{r},\ldots,h(d)+\frac{r-1}{r}$ in the $[0,1)$-ring (recall that $|I_i|=c_i\cdot s$). However, in what follows in the next section, we aim at yielding redundancy, fairness and adaptivity for disks up to size $\leq\frac{1}{r}$.

In [SS05], a related strategy, called *DHHT*, was introduced but, similar to SHARE and SIEVE, it has not been shown yet that this strategy can handle both redundancy and fairness simultaneously and in an appropriate manner.

Recently, Brinkmann et al. [BEMadHS07] proposed the first scheme for systems of disks with arbitrary capacities that is fair and redundant but what is only $\Theta(r^2)$-adaptive in the worst case for r copies per data item whereas we are aiming at a constant adaptivity that is independent of r.

3.5.2 The SPREAD Strategy

For the rest of this chapter, we describe and analyze the SPREAD strategy in more detail. First, we sketch the basic framework of SPREAD but leaving out various details about how to achieve fairness, redundancy and adaptivity. Afterwards, we bound the time- and space-efficiency of SPREAD (Section 3.5.2.2), describe how to achieve fairness and redundancy (Section 3.5.2.3), and show how to make SPREAD $O(1)$-adaptive (Sections 3.5.2.4 and 3.5.2.5). At the end, we discuss how to extend SPREAD to the case that the data items have different levels of redundancy.

3.5.2.1 The basic scheme.

Like in the previous sections, in the following, we again consider a given storage network $S(n,C[n])$ and a corresponding share tuple $\bar{c}$ for the nodes in the set $[n]$. Again, assume the nodes to be numbered uniquely in a range from 1 to n, and furthermore due to the adaptivity condition, for any history of changes, we always use the strategy that every node newly entering the system obtains the lowest available identifier ≥ 1, thus, we can make sure that the nodes will be numbered in the defined range from 1 to n any further (note that the capacity of any currently unoccupied identifier is equal to 0, but it will nevertheless be part of the SPREAD data structure.).

SPREAD uses three (pseudo-)random ($O(\log n)$-universal) hash functions: a hash function $h : V \to [0,1)$ for the nodes and two hash functions $g_1, g_2 : U \to [0,1)$ to map the data items into the $[0,1)$-interval (which again will be considered as a modulo1 ring). Similar to the SHARE strategy, SPREAD makes use of a stretch factor $s = \sigma \cdot \log N$, where $N = |V|$ (resp. the maximum number of nodes the system can expect) and $\sigma \geq 1$ is a sufficiently large constant such that $s \in \mathbb{N}$. Note that, while in SHARE, the stretch factor was used to ensure the complete coverage of the unit ring, in SPREAD, it is furthermore used to guarantee that for each point $x \in [0,1)$, there are at least r intervals passing x. Now, given these preconditions, the SPREAD strategy works as follows.

For every node $v \in [n]$, we identify an interval $I(v) = [h(v), h(v) + s \cdot r \cdot c_i \mod 1)$ of length $s \cdot r \cdot c_i$ in the $[0,1)$-ring, where $h(v)$ is called the *starting point* of the interval and $h(v) + s \cdot r \cdot c_i \mod 1$ is its *endpoint*. If $s \cdot r \cdot c_i > 1$, we consider $I(v)$ to be wrapped around the whole $[0,1)$-interval $\lfloor s \cdot r \cdot c_i \rfloor$ many times.

With respect to the previous strategy, the SPREAD scheme realizes some form of a hybrid solution combining the Consistent Hashing approach and the SHARE strategy. In particular, like Consistent Hashing, the SPREAD data structure maintains a partition of the $[0,1)$ interval into *n frames* $F_1, \ldots, F_n$ with F_v starting at $h(v)$ and ending at the closest successor of $h(v)$ among the points $\{h(1), \ldots, h(n)\} \setminus \{h(v)\}$ (recall that $[0,1)$ is treated as a ring). Surely, the intervals differ in size, and since all intervals are allocated to the $[0,1)$-ring independently and uniform at random, they determine overlapping areas, called

areas of influence, for some given point $x \in [0,1)$, that is, there are numerous intervals passing x (see Figure 3.6 for an example).

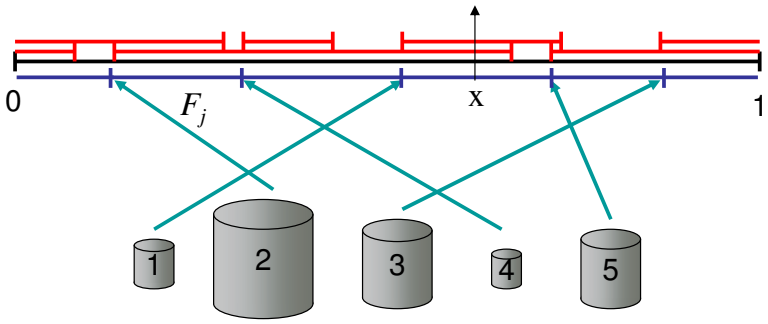

Figure 3.6: The interval decomposition of SPREAD

Moreover, like in SHARE, for reducing the non-uniform allocation problem to a uniform one, each frame F_v is further partitioned into a number of *subframes* (we will explain later how these subframes are chosen. For now, just note that the subframe decomposition only depends on n and $h(1),\ldots,h(n)$ and not on the capacities of the nodes, and as n grows according to system scalings, some of these subframes may be partitioned into smaller subframes). Then, given the replication parameter r, for each subframe f, SPREAD aims at maintaining one (and sometimes two, as will be explained later) $(\frac{s}{\alpha}) \times r$-table T_f consisting of $r \cdot \frac{s}{\alpha}$ slots which are organized into $\frac{s}{\alpha}$ *groups* (respectively columns) of size r each, where $\alpha > 0$ is a small constant that is selected such that a certain degree of fairness is maintained. Every slot is assigned to (resp. *owned* by) exactly one node, and the assignment has to be chosen so that for each group of r slots, each slot is assigned to a different node (see Figure 3.7). Then, we can use the following strategy for ensuring redundancy.

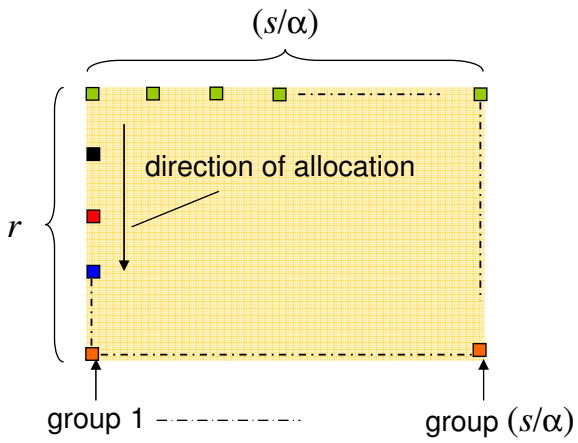

Figure 3.7: Allocation table for a subframe f

At first, whenever we want to read or overwrite the copies of some data item d, we perform the following steps.

1. Identify the unique subframe f via $g_1(d) \in f$,
2. Pick group $\lceil (\frac{s}{\alpha}) \cdot g_2(d) \rceil$ in T_f and either read or store the copies of d in the r nodes owning the slots in this group

(in Figure 3.8, the allocation of a data item to the slots in the corresponding subframe table is depicted). Given that the hash functions can be evaluated in constant time and there are m subframes in the system, standard data structures such as search trees and arrays can be used to obtain the following result (recall that we assign slots to nodes, and surely, n different nodes require $\log n$ bits for encoding).

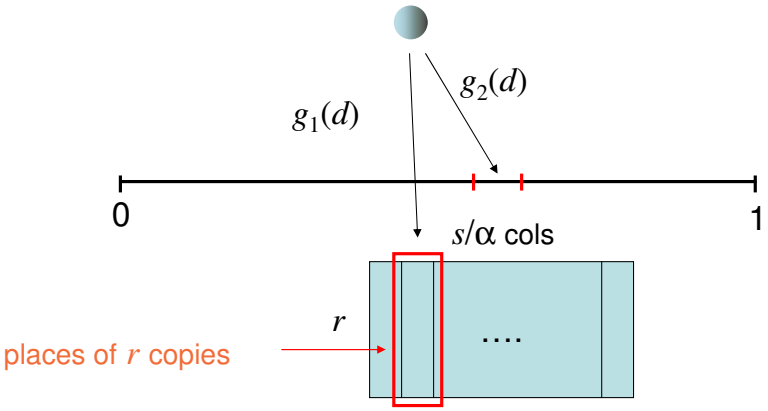

Figure 3.8: Allocation of a slot in a subframe table

Lemma 3.42. *The lookup and insert operations of SPREAD can be implemented with runtime* $O(r + \log m)$ *and space* $O(m\,(r \cdot \frac{s}{\alpha})\log n)$.

Hence, as long as m is close to linear in n, SPREAD is time- and space-efficient. In the following subsections, we show that there is a way of maintaining the tables so that also the fairness, redundancy and adaptivity conditions are satisfied.

3.5.2.2 Subframe management.

As we see in the following, to make the reduction phase that transforms the non-uniform problem into a number of uniform ones be able to cope also with a scaling storage system appropriately, ideally, we would like to have the following subframe decomposition for each frame $F_v \subseteq [0,1)$. For the subframes $f_0, f_1, f_2, \ldots$ following F_v in the $[0,1)$-ring (in ascending direction), it holds that

$$|f_i| = \varepsilon(1+\varepsilon)^i \cdot |F_v| \quad \text{for some fixed } \varepsilon > 0. \tag{3.15}$$

If this were true, we could approximate any interval $I(v) \subseteq [0,1)$ of size $|I(v)| \geq |F_v|$ by an interval $I'(v)$ ending at the starting point of some subframe f_i so that $|I'(v)|$ is within $(1 \pm \varepsilon)|I(v)|$. In fact, the following lemma holds.

Lemma 3.43. *Suppose some number $k \in \mathbb{N}$ to be the maximum value for which it holds that $|F_v| + \sum_{i=0}^{k-1} |f_i| \leq |I(v)|$. Then, it also holds that $|F_v| + \sum_{i=0}^{k-1} |f_i| \geq (1-\varepsilon)|I(v)|$ and $|F_v| + \sum_{i=0}^{k} |f_i| \leq (1+\varepsilon)|I(v)|$.*

Proof. It holds that

$$|F_v| + \sum_{i=0}^{k-1} |f_i| = (1+\varepsilon \sum_{i=0}^{k-1} (1+\varepsilon)^i)|F_v| = (1+\varepsilon)^k |F_v|$$

for any $k \geq 0$. Hence, for $|F_v| + \sum_{i=0}^{k-1} |f_i| \leq |I(v)| \leq |F_v| + \sum_{i=0}^{k} |f_i|$, we get that

$$|F_v| + \sum_{i=0}^{k-1} |f_i| \geq \frac{1}{1+\varepsilon}|I(v)| = (1 - \frac{\varepsilon}{1+\varepsilon})|I(v)| \geq (1-\varepsilon)|I(v)|$$

and $|F_v| + \sum_{i=0}^{k} |f_i| \leq (1+\varepsilon)|I(v)|$. ■

Thus, we can either decide to round $I(v)$ down to the starting point of the subframe containing its endpoint or up to the starting point of the next subframe in order to obtain an interval $I'(v)$ whose size is within $(1 \pm \varepsilon)|I(v)|$. If $|I(v)|$ is at most $|F_v|$, we identify $I'(v)$ with $I(v)$, i.e., we do not round $I(v)$ (see Figure 3.9 for illustration).

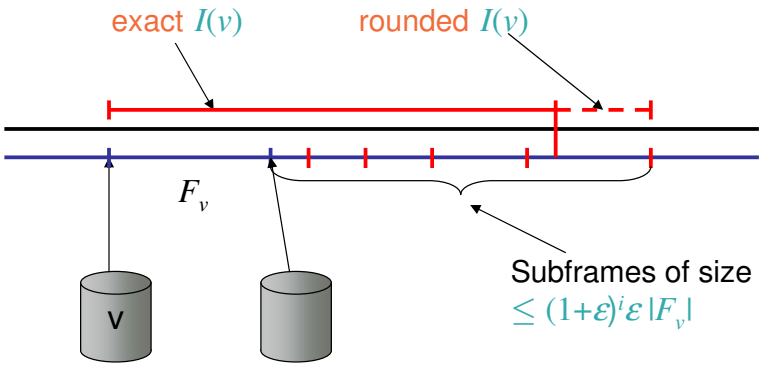

Figure 3.9: Rounded Up Interval

Of course, choosing a subframe decomposition such that for each frame F_v, there are subsequent subframes $f_0, f_1, f_2, \ldots$ of sizes $|f_i| = \varepsilon(1+\varepsilon)^i |F_v|$ is not possible, but it is sufficient to cut each frame F_w into subframes such that the following condition is true for every frame F_v:

Subframe condition: For any two frames or subframes f and f', let $\Delta(f,f')$ be the distance (in ascending direction along the $[0,1)$-ring) from the endpoint of f to the starting point of f'. Furthermore, for every $w \in [n] \setminus \{v\}$ and every subframe f in F_v, we require that $|f| \leq \varepsilon(1+\varepsilon)^k |F_w|$, where $k \in \mathbb{N}_0$ is maximum possible so that

$$\varepsilon \sum_{i=1}^{k-1} (1+\varepsilon)^i |F_w| \;\leq\; \Delta(F_w, f).$$

If this condition is true, it is easy to see that the interval $I(v)$ can be rounded down to the starting point of the subframe f_i containing its endpoint or up to the starting point of the next subframe f_{i+1} in order to obtain an interval $I'(v)$ with $|I(v')| \leq (1 \pm \varepsilon)|I(v)|$. Now, to satisfy the subframe condition, we use the following decomposition strategy for the $[0,1)$-ring.

Given $h(1), \ldots, h(n) \in [0,1)$, we start with some single subframe representing the entire $[0,1)$-interval, and we keep cutting subframes in half until the subframe condition is true everywhere (when considering each subframe crossing $b \geq 1$ frame borders as being cut into $b+1$ subframes at these borders). This leads to a unique decomposition into subframes (for any given node mappings $h(1), \ldots, h(n)$) with the property that for every subframe f that is not the first or last subframe in some frame F_v, $|f| = \frac{1}{2^k}$ for some $k \in \mathbb{N}_0$. Moreover, whenever the number n of disks is increased by system scaling, the decomposition strategy will only cause some subframes to be cut into smaller subframes (which turns out to be useful for the adaptivity of SPREAD). The following two lemmas show that, w.h.p., our decomposition rule does not create too many subframes.

Lemma 3.44. *The number of frames F_v in the system with $|F_v| \leq \frac{\delta}{n}$ for some given $\delta > 0$ is bounded by $2\delta n + O(\sqrt{n})$, w.h.p. Furthermore, w.h.p. there is no frame F_v with $|F_v| \leq \frac{1}{n^k}$ for some constant k, w.h.p.*

Proof. For every node $v \in [n]$, consider a random variable X_v with $X_v = h(v)$. Then, given a fixed δ, let the function

$$f(X_1, \ldots, X_n) \;:\; [0,1)^n \to \mathbb{N}$$

represent the number of frames F_v with $|F_v| \leq \frac{\delta}{n}$. Certainly, whenever $(x_1, \ldots, x_n)$ and $(x'_1, \ldots, x'_n)$ differ in at most one coordinate, it holds that

$$|f(x_1, \ldots, x_n) - f(x'_1, \ldots, x'_n)| \;\leq\; 2.$$

Moreover, for any node $v \in [n]$ and $\delta \leq 1$, it holds that

$$\begin{aligned}\Pr\left[|F_v| \leq \frac{\delta}{n}\right] &= 1-\left(1-\frac{\delta}{n}\right)^{n-1} \\ &\leq 1-\left(1-(n-1)\frac{\delta}{n}+\binom{n-1}{2}\left(\frac{\delta}{n}\right)^2\right) \\ &= (n-1)\frac{\delta}{n}-\binom{n-1}{2}\left(\frac{\delta}{n}\right)^2 \leq \delta\end{aligned}$$

which implies that $\mathrm{E}[f] \leq \delta n$. Hence, it follows from the method of bounded differences (c.f. Section 3.1.3.1) that for any $d \geq 0$, it holds that

$$\Pr[f \geq \delta n + d] \leq e^{-d^2/(2n)}.$$

Choosing $d = \delta n + \Theta(\sqrt{n})$ results in the first part of the lemma. The second part holds because for any node v, it holds that

$$\Pr\left[|F_v| \leq \frac{1}{n^k}\right] \leq \frac{1}{n^{k-1}}.$$

Therefore, the probability that there exists a $v \in [n]$ with $|F_v| \leq \frac{1}{n^k}$ is polynomially small in n. ■

Lemma 3.45. *The number of subframes in the system is bounded by $O(\frac{n}{\varepsilon})$, w.h.p.*

Proof. For any $i \in \mathbb{N}_0$, let S_i be the set of all frames of size in $[\frac{2^{-i}}{n}, \frac{2^{-(i+1)}}{n})$. Then, for each frame $F \in S_i$, we need at most

$$k = \left\lceil \log_{1+\varepsilon}\left(\frac{1}{n|F|}\right)\right\rceil \leq \lceil \log_{1+\varepsilon} 2^{i+1}\rceil = O\left(\frac{i}{\varepsilon}\right)$$

subframes $f_0, f_1, \ldots, f_k$ in the ideal setting until $|f_k| = \varepsilon(1+\varepsilon)^k|F| \geq \frac{\varepsilon}{n}$. Hence, it follows from Lemma 3.44 that the total number of subframes needed is bounded by

$$\sum_{i=0}^{\log n}\left(\frac{2n}{2^i}\right)\cdot O\left(\frac{i}{\varepsilon}\right) + O(\sqrt{n})\cdot O\left(\frac{\log n}{\varepsilon}\right) + O\left(\frac{n}{\varepsilon}\right) = O\left(\frac{n}{\varepsilon}\right)$$

w.h.p. ■

Next, we explain how to manage the tables.

3.5.2.3 Table management.

For each subframe $f \subseteq [0,1)$, $C(f)$ represents a multiset of nodes w for which $f \cap I(w) \neq \emptyset$ holds. Furthermore, for every node w in $C(f)$, let its *multiplicity* $\mu_f(w)$ be defined as the number of times w occurs in $C(f)$ (see Section 3.1.3.4 for details on multisets). Initially, $\mu_f(w)$ is set to the number of times $I(w)$ crosses the starting point of f, thus, surely, $\mu_f(w) \in \{\lfloor r \cdot s \cdot c_w \rfloor, \lceil r \cdot s \cdot c_w \rceil\}$. Moreover, since $c_i \leq \frac{1}{r}$ for every $i \in [n]$, it holds that $\mu_f(w) \in \{0, \ldots, s\}$.

Now, for what follows, consider some frame F_v. Then, for every subframe $f \subseteq F_v$ and nodes $w \neq v$, we are using the following rules:

Rounding conditions:

1. Whenever the endpoint of $I(w)$ crosses f completely (i.e. moves beyond the start- and endpoint of f) for the first time, we set $\mu_f(w) = \mu_f(w) + 1$.
2. Whenever the endpoint of $I(w)$ moves below the starting point of f for the first time after crossing the endpoint of f (or after w has been introduced to the system), we set $\mu_f(w) = \mu_f(w) - 1$.

We also apply the same rules if $w = v$ and $|I(v)| > |F_v|$. For the special case that $w = v$ and $|I(v)| \leq |F_v|$, $\mu_f(w)$ is defined as the number of times $I(w)$ crosses the starting point of f (recall that the frames F_v are defined by the values $h(v)$, thus, $|I(v)| \leq F_v$ may occur).

With these rules, it holds that whenever the endpoint of $I(w)$ is outside of f, then $\mu_f(w)$ equals the number of times $I(w)$ crosses f, and otherwise $\mu_f(w) \in \{\lfloor r \cdot s \cdot c_i \rfloor, \lceil r \cdot s \cdot c_i \rceil\}$. Hence, since no interval can start inside of f, the number of times intervals cross f at the left and right endpoints is an upper and lower bound on $|C(f)|$. More precisely, for any subframe f, we can bound $|C(f)|$ as follows.

Lemma 3.46. *For every subframe $f \subseteq [0,1)$ and corresponding multiset $C(f)$, $|C(f)|$ is within $(1 \pm \beta) r \cdot s$, w.h.p., where the constant $\beta > 0$ can be made arbitrarily small depending on the constant σ in s.*

Proof. To bound $|C(f)|$ for any subframe $f \subseteq [0,1)$, the arguments above imply that it suffices to consider any point $x \in [0,1)$ that represents a starting or endpoint of some interval $I(v)$. In the following, let us first consider the starting point of an interval $I(v)$.

Consider some fixed node v with starting point $h(v) \in [0,1)$. For any node w in the system (including v), let the random variable X_w be defined as

$$X_w = \lfloor r \cdot s \cdot c_w \rfloor + Y_w$$

where the binary random variable $Y_w = 1$ if and only if $I(w)$ contains $h(v)$ $\lceil r \cdot s \cdot c_w \rceil$ many times. Furthermore, let $X = \sum_w X_w$ and $Y = \sum_w Y_w$. It certainly holds that $\mathrm{E}[X_w] = s \cdot r \cdot c_w$ for all $w \neq v$ and that $X = \lceil s \cdot r \cdot c_v \rceil + \sum_{w \neq v} X_w$. Hence,

$$\mathrm{E}[X] = \lceil s \cdot r \cdot c_v \rceil + s \cdot r \cdot (1 - c_v) \in [s \cdot r,\ s \cdot r + 1].$$

Moreover, $\mathrm{E}[Y] \leq \mathrm{E}[X]$, and since the starting points of the intervals are chosen independently at random and the Y_w's are binary random variables, it follows from the Chernoff bounds (see Section 3.1.3.1) that

$$\Pr[|Y - \mathrm{E}[Y]| \geq \beta \mathrm{E}[Y]] \;\leq\; e^{-\beta^2 \frac{\mathrm{E}[Y]}{2(1+\beta/3)}}$$

for any $\beta > 0$. Hence, X is within $(1 \pm \beta) s \cdot r$, w.h.p., where the constant $\beta > 0$ can be made arbitrarily small depending on the constant σ in s.

For the endpoint of an interval $I(v)$, it is easy to see that $\mathrm{E}[X] \in [s \cdot r - 1,\ s \cdot r]$. Thus, the same deviation bounds can also be shown here. ■

For our further descriptions, we need somewhat more refined estimates on the sizes of multisets corresponding to the nodes in the system. As we will see later, this is important for the design and management of the used tables to appropriately handle system dynamics.

For what follows, consider the sets

$$W_\ell := \left\{ v \in [n] \mid c_v \geq \tfrac{1}{(s-1)r} \right\} \;\subseteq\; [n]$$

and $W_s := [n] \setminus W_\ell$ (it is easy to see that all nodes $v \in W_\ell$ are guaranteed to have multiplicity $\mu_f(v) \geq 1$ in every subframe $f \subseteq [0,1)$).

Now, for any subframe f, we define $C_\ell(f) := C(f)|_{W_\ell}$ (i.e., the multiset containing only those nodes in $C(f)$ that are also in W_ℓ) and $C_s(f) := C(f)|_{W_s}$. Furthermore, let $a_\ell = \sum_{v \in W_\ell} c_v$ and $a_s = 1 - a_\ell$. Then, a more refined version of Lemma 3.46 is as follows.

Lemma 3.47. *For any subframe f and any subset $W'_\ell \subseteq W_\ell$, it holds that $|C'_\ell(f)|$ is within $s \cdot r \cdot a'_\ell \pm \beta |W'_\ell| + O(\log n)$, w.h.p., where $C'_\ell(f) = C_\ell(f)|_{W'_\ell}$ and $a'_\ell = \sum_{v \in W'_\ell} c_v$. Furthermore, $|C_s(f)|$ is within $(1 \pm \beta) s \cdot r \cdot a_s + O(\log n)$, w.h.p.*
In both cases, the constant $\beta > 0$ can be made arbitrarily small depending on the constant σ in s.

Proof. Recall the definition of X_w and Y_w in the proof of the previous lemma. Applied to nodes $w \in W'_\ell$, it holds that

$$\mathrm{E}[X] = \sum_{w \in W'_\ell} \lfloor s \cdot r \cdot c_w \rfloor + \mathrm{E}[Y] \in [s \cdot r \cdot a'_\ell - 1,\ s \cdot r \cdot a'_\ell + 1].$$

Since $\mathrm{E}[Y] \leq |W'_\ell|$, the first bound follows from the fact that

$$\Pr[|Y - \mathrm{E}[Y]| \geq \beta \mathrm{E}[Y]] \;\leq\; e^{-\beta^2 \frac{\mathrm{E}[Y]}{2(1+\beta/3)}}$$

for any $\beta > 0$.

The second bound follows along the same lines as in the proof of Lemma 3.46. ■

As we know now, any subframe is always crossed by a sufficient number of intervals. Furthermore, the following result is crucial for SPREAD to be redundant.

Lemma 3.48. *For every subframe f, there are at least r different nodes in $C(f)$, w.h.p., where the probability depends on the constant σ in the stretch factor.*

Proof. Let $q = |W_\ell|$. If $q \geq r$, then the lemma is trivially true. So suppose that $q < r$. Since $c_v \leq \frac{1}{r}$ for all $v \in [n]$, a relative capacity of at least $1 - \frac{q}{r}$ must be covered by W_s. Hence,

$$|W_s| \geq \left(1 - \tfrac{q}{r}\right) / \left(\tfrac{1}{(s-1)r}\right) = (s-1)(r-q)$$

and

$$\sum_{v \in W_s} |I(v)| \geq (r \cdot s)\left(1 - \frac{q}{r}\right) = s(r-q).$$

Suppose that the nodes in W_s are numbered from 1 to t. Consider any fixed node $v \in W_s$ and focus on the point y implied by v's interval $I(v) = [x, y)$. For any node $w \in W$, let the binary random variable $X_w = 1$ if and only if $y \in I'(w)$ and let $X = \sum_w X_w$. Since

$$\mathrm{E}[X_w] = \Pr[X_w = 1] = \min\{|I'(w)|, 1\} \text{ for all } w \in W_s \setminus \{v\},$$

it holds that

$$\begin{aligned} \mathrm{E}[X] &= q + \sum_{w \in W_s \setminus \{v\}} \mathrm{E}[X_w] = q + \sum_{w \in W_s \setminus \{v\}} \min\{|I'(w)|, 1\} \\ &\geq q + (1-\varepsilon)s(r-q) - 1 \geq r + (1-\varepsilon)s - 2 \end{aligned}$$

Since $s = \Theta(\log N)$ and the starting points of the intervals are chosen uniformly and independently at random, it follows from the Chernoff bounds that $X \geq r$ for point y, w.h.p. Since we only need to consider those points in $[0,1)$ that are starting points or endpoints of intervals $I(v)$ in order to cover all subframes in $[0,1)$ and the lowest values for $\mathrm{E}[X]$ are reached when focusing on endpoints of intervals $I(v)$ of nodes $v \in W_s$, the lemma follows. ■

Recall that for each subframe f, we maintain one (and sometimes two) $\frac{s}{\alpha} \times r$-table T_f of $r \cdot \frac{s}{\alpha}$ slots which are organized into $\frac{s}{\alpha}$ *groups* (or columns) of size r each (Figure 3.7). We assume that $\alpha > 0$ is a sufficiently small constant with $\frac{1}{\alpha} \in \mathbb{N}$. Our goal is to assign the slots of T_f to nodes in f so that the following conditions are met:

Table conditions:

1. Every slot is assigned to (resp. *owned* by) exactly one node in $C(f)$.

2. Every node v in $C(f)$ owns within $(1 \pm \gamma)\mu_f(v)/\alpha$ many slots but at most $\frac{s}{\alpha}$ many slots in T_f for some constant $0 < \gamma < 1$ to be specified below.

3. Every group consists of slots belonging to different nodes.

The constant γ which is sufficient to maintain the table conditions is given in the following lemma. Moreover, in this lemma, α is the parameter used in the table size and β is the parameter used in the Lemmas 3.46 and 3.47.

Lemma 3.49. *If the bounds in Lemma 3.47 are true and α and β are sufficiently small constants, then Conditions 1 and 2 can be met with any $\gamma \geq \frac{\beta}{1-\beta} + \alpha$.*

Proof. Recall the bounds in Lemma 3.47 and ignore the $O(\log n)$ terms for a moment. Then, in this proof, we consider the nodes in W_ℓ and W_s separately, starting with W_ℓ.

Let $W_h \subseteq W_\ell$ be the subset of all nodes $v \in W_\ell$ with capacity c_v large enough so that it is guaranteed that $\mu_f(v) \geq \frac{1}{\gamma}$ for γ as chosen in the lemma. In this case, every $v \in W_h$ satisfies

$$(1-\gamma)\,\frac{\mu_f(v)}{\alpha} \;\leq\; \frac{\mu_f(v)-1}{\alpha} \;\leq\; \left\lfloor s \cdot r \cdot \frac{c_v}{\alpha} \right\rfloor$$

and

$$(1+\gamma)\,\frac{\mu_f(v)}{\alpha} \;\geq\; \frac{\mu_f(v)+1}{\alpha} \;\geq\; \left\lceil s \cdot r \cdot \frac{c_v}{\alpha} \right\rceil.$$

Moreover, $\lceil s \cdot r \cdot \frac{c_v}{\alpha} \rceil \leq \frac{s}{\alpha}$. Hence, each $v \in W_h$ can be given either $\lfloor s \cdot r \cdot \frac{c_v}{\alpha} \rfloor$ or $\lceil s \cdot r \cdot \frac{c_v}{\alpha} \rceil$ many slots without violating Condition 1 implying a slot assignment to the nodes in W_h such that the total number of slots used by the nodes in W_h is in

$$\left[\left\lfloor s \cdot r \cdot \frac{a_h}{\alpha} \right\rfloor, \left\lceil s \cdot r \cdot \frac{a_h}{\alpha} \right\rceil \right]$$

where $a_h = \sum_{v \in W_h} c_v$. This is perfect up to an additive 1.

Next, consider the remaining nodes in the subset $W'_\ell = W_\ell \setminus W_h$ with capacity c_v such that $\mu_f(v) < \frac{1}{\gamma}$. For this, let $C'_\ell(f) = C_\ell(f)|_{W'_\ell}$ be the multiset of nodes $v \in C_\ell(f)$ with $v \notin W_h$. In this case, for each node $v \in W'_\ell$, it holds that

$$(1+\gamma)\,\frac{\mu_f(v)}{\alpha} \;<\; \frac{s}{\alpha}$$

(given that s is sufficiently large). Thus, we do not have to worry about limiting the number $\frac{s}{\alpha}$ for the slots of v.

Let $a'_\ell = \sum_{v \in W'_\ell} c_v$. Then, suppose for an upper bound on the number of slots per node that $|C'_\ell(f)| = s \cdot r \cdot a'_\ell - \beta |W'_\ell|$ (see Lemma 3.47). If a constant $\phi > 0$ is chosen such that

$$\sum_{v \in W'_\ell} \left\lceil \frac{\mu_f(v)}{\alpha} + \frac{\phi}{\alpha} \right\rceil \;\geq\; s \cdot r \cdot \frac{a'_\ell}{\alpha}, \tag{3.16}$$

then it suffices to assign at most

$$\frac{\mu_f(v)+\phi}{\alpha}+1$$

slots to every node $v \in C'_\ell(f)$ to cover at least an a'_ℓ-fraction of the slots in f. Furthermore, since $\sum_{v\in W'_\ell} \mu_f(v) = |C'_\ell(f)|$, Inequality 3.16 above holds if and only if

$$|C'_\ell(f)| \;\geq\; s\cdot r\cdot a'_\ell - \phi|W'_\ell|,$$

so we have to choose $\phi \geq \beta$ for this.

For a lower bound, suppose that $|C'_\ell(f)| = s\cdot r\cdot a'_\ell + \beta|W'_\ell|$. If the constant $\phi > 0$ is chosen so that

$$\sum_{v\in W'_\ell}\left[\frac{\mu_f(v)}{\alpha}-\frac{\phi}{\alpha}\right] \;\leq\; s\cdot r\cdot\frac{a'_\ell}{\alpha},$$

then at least

$$\frac{\mu_f(v)-\phi}{\alpha}-1$$

slots can be assigned to every node $v \in C'_\ell(f)$ to cover at most an a'_ℓ-fraction of the slots in f. For this, we have to choose $\phi \geq \beta$. Together with the fact that $\mu_f(v) \geq 1$ for all $v \in C'_\ell(f)$, it follows that $\gamma = \beta + \alpha$ is sufficient so that

$$\frac{\mu_f(v)+\beta}{\alpha}+1 \leq (1+\gamma)\,\frac{\mu_f(v)}{\alpha}$$

and

$$\frac{\mu_f(v)-\beta}{\alpha}-1 \geq\; (1-\gamma)\,\frac{\mu_f(v)}{\alpha}.$$

Thus, we are done with determining bounds on the slot assignment for the nodes in W_ℓ, that is, for nodes v that are guaranteed to have multiplicity $\mu_f(v) \geq 1$ in $C(f)$.

Finally, we consider the nodes in W_s, that is, nodes v with $\mu_f(v) < 1$ that may not cross every subframe $f \in [0,1)$. Suppose for an upper bound on the number of slots per node that $|C_s(f)| = (1-\beta)s\cdot r\cdot a_s$ (see Lemma 3.47). If then constant $\phi > 0$ is chosen so that

$$\sum_{v\in W_s}(1+\phi)\,\frac{\mu_f(v)}{\alpha} \;\geq\; s\cdot r\cdot\frac{a_s}{\alpha},$$

it suffices to assign at most

$$(1+\phi)\,\frac{\mu_f(v)}{\alpha}+1$$

slots to every node $v \in C_f$ to cover an a_s-fraction of the slots in f. Since it holds that $\sum_{v\in W_s}\mu_f(v) = |C_s(f)|$, we have to choose ϕ such that

$$(1+\phi)(1-\beta)\,s\cdot r\cdot\frac{a_s}{\alpha} \;\geq\; s\cdot r\cdot\frac{a_s}{\alpha},$$

which works for $\phi \geq \frac{1}{1-\beta} - 1 = \frac{\beta}{1-\beta}$.

For a lower bound, suppose that $|C_s(f)| = (1+\beta)s \cdot r \cdot a_s$. If then the constant $\phi > 0$ is chosen so that

$$\sum_{v \in W_s} (1-\phi)\frac{\mu_f(v)}{\alpha} \;\leq\; s \cdot r \cdot \frac{a_s}{\alpha},$$

at least

$$(1-\phi)\frac{\mu_f(v)}{\alpha} - 1$$

slots can be assigned to every node $v \in C_s(f)$ without exceeding an a_s-fraction of the slots in f. For this, we have to choose ϕ such that

$$(1-\phi)(1+\beta)\, s \cdot r \cdot \frac{a_s}{\alpha} \;\leq\; s \cdot r \cdot \frac{a_s}{\alpha}$$

which works for $\phi \geq 1 - \frac{1}{1+\beta} = \frac{\beta}{1+\beta}$. Hence, $\gamma \geq \frac{\beta}{1-\beta} + \alpha$ is sufficient for both cases.

Finally, notice that there is this $O(\log n)$ term in the bounds in Lemma 3.47 that we ignored so far. However, whenever this term is dominant for a set of nodes in some subframe f_j, then it holds that $\sum_{v \in f_j} c_v \leq \delta$ for some constant $\delta > 0$ that can be made arbitrarily small depending on the constant σ in the stretch factor s. In other words, since we are considering only three sets of nodes (W_ℓ, W'_ℓ, W_s), those sets of nodes where the $O(\log n)$ term is negligible (and can therefore be covered by β) represent a total capacity of at least $1 - 2\delta$ which is sufficient to fill all slots just with these nodes, or leave enough space for the other nodes, as long as δ is sufficiently small compared to α and β. ■

If Conditions 1 and 2 are true, also Condition 3 can be met. To show this, consider the slots to be numbered column-wise from 1 to $r \cdot \frac{s}{\alpha}$ by giving slot (i, j) in group i the number $(j-1) \cdot \frac{s}{\alpha} + i$ (c.f. Figure 3.7). Then, assign the slots to the nodes such that each node $v \in C(f)$ owns a consecutive sequence of slots. As every node owns at most $\frac{s}{\alpha}$ slots, no group can have two slots owned by the same node, which proves our claim.

However, the challenge, of course, will be to maintain these conditions in case of system changes caused by some change operation ω without rearranging too many slot assignments. Before we show how to do this, we prove that the given table conditions allow us to maintain fairness.

Lemma 3.50. *If the table conditions are met, then for every node $v \in [n]$ in the system and every data item $d \in U$, it holds that*

$$\Pr[v \text{ stores a copy of } d] \in [(1-\gamma)(1-\varepsilon)\, r \cdot c_v,\; (1+\gamma)(1+\varepsilon)\, r \cdot c_v]$$

for the $\gamma > 0$ as given in Condition 2 and $\varepsilon > 0$ as chosen for the interval rounding.

Proof. Consider any node $v \in [n]$ and data item $d \in U$. Let $p = |I'(v)| \mod 1$, where $I'(v)$ is the rounded form of $I(v)$ (see Figure 3.9). For an area $A_p = [x_i, x_j] \subseteq [0,1)$ of size $|A_p| = p$, the multiplicity of v is $\mu_f(v) = \lceil |I'(v)| \rceil$ for all subframes $f \subseteq A_p$. For the remaining area $A'_p \subseteq [0,1)$ of size $|A'_p| = (1-p)$, the multiplicity is $\mu_f(v) = \lfloor |I'(v)| \rfloor$ for all subframes $f \subseteq A'_p$. Moreover, it follows from the table conditions that for any node v in some subframe f,

$$\Pr[v \text{ selected by a data item}] \in \left[(1-\gamma)\,\frac{\mu_f(v)}{s}\,,\ \frac{\min\{(1+\gamma)\,\mu_f(v), s\}}{s} \right].$$

Hence, it holds for data item d that

$$\begin{aligned} \Pr[v \text{ stores a copy of } d] &\geq p \cdot \frac{(1-\gamma)\lceil |I'(v)| \rceil}{s} + (1-p) \cdot \frac{(1-\gamma)\lfloor |I'(v)| \rfloor}{s} \\ &= (1-\gamma) \cdot \frac{|I'(v)|}{s} \geq (1-\gamma)(1-\varepsilon)\, r \cdot c_v \end{aligned}$$

Similarly, it holds that $\Pr[v \text{ stores a copy of } d] \leq (1+\gamma)(1+\varepsilon)\, r \cdot c_v$, which implies the lemma. ■

Since $\varepsilon > 0$ and $\gamma > 0$ can be made arbitrarily small, the lemma implies the fairness of SPREAD. Next we show that SPREAD is also adaptive.

3.5.2.4 Amortized adaptivity.

We start with amortized adaptivity, i.e. we show how to perform updates so that movements of data copies can always be charged to capacity changes in the past.

Note that SPREAD does not need to perform any adaptations of the slots as new data items are added or old data items are removed from the system since Lemma 3.50 implies that the data items in the system will remain fairly distributed among the nodes. Hence, it remains to describe how to react to changes in the capacities of the nodes.

In what follows, we suppose that the capacities of the nodes change from $(c_1, \ldots, c_n)$ to $(c'_1, \ldots, c'_n)$. In case that the number n of nodes in the system increases, then, for each node identifier $v \in V$ that has not been used before, we have to rearrange the subframe layout in the $[0,1)$-interval in a way that some subframes may have to be cut into smaller subframes that take over the tables of the previous subframes. Then, it follows that if the previous subframes satisfied the table conditions, the new ones will also do so. After that, our rounding conditions may require updates to these tables, which can be charged to past capacity changes as for the other cases below. At last, we adapt the intervals $I(v)$ to the new capacity distribution $(c'_1, \ldots, c'_n)$. Clearly, this may cause changes in the multiset $C(f)$ of some subframe f inducing required updates in its table (or tables). We call a subframe f *dirty* if $f \subseteq F_v$, and furthermore, f contains the endpoint of the interval $I(v)$ with $|I(v)| \leq |F_v|$. Otherwise, it is called *clean*.

In the following, we first describe how to update the table of a clean subframe, and then we consider dirty subframes.

Updating a clean subframe f. Let $C(f)$ be the multiset of nodes in f before the change and $C'(f)$ be the multiset of nodes in f after the change in capacities. Furthermore, let $\mu_f^C(v)$ denote the multiplicity of some node $v \in C(f)$ and $\mu_f^{C'}(v)$ the multiplicity of $v \in C'(f)$. If then $C'(f) \neq C(f)$, we go through the following stages.

1. **Pairing stage:** Suppose that

$$\sum_{v \in C(f)} (\mu_f^C(v) - \mu_f^{C'}(v)) = \delta_d$$

is the total decrease in the multiplicities of nodes in $C(f)$, and

$$\sum_{v \in C(f)} (\mu_f^{C'}(v) - \mu_f^C(v)) = \delta_i$$

is the total increase in the multiplicities of nodes in $C(f)$. Then, we can identify $|\delta_i - \delta_d|$ pairs of nodes (v,w) where v wants to decrease its multiplicity whereas w wants to increase its multiplicity. For each such pair, we set

$$\mu_f^{C'}(v) := \mu_f^C(v) - 1 \quad \text{and} \quad \mu_f^{C'}(w) := \mu_f^C(w) + 1$$

and then change slot assignments according to $\mu_f^{C'}(v)$ and $\mu_f^{C'}(w)$, respectively, until table Condition 2 is satisfied for v and w.

For each such slot reassignment, we distinguish between three cases.

a) If both v and w violate Condition 2, then a slot of v is given to w.

b) If only v violates Condition 2, we give a slot of v to any node w' who can still take a slot without violating Condition 2 (we will see below that such a node w' can always be found, w.h.p.).

c) If only w violates Condition 2, we give a slot from any node v' who can lose a slot without violating Condition 2 to w.

For each slot $x \in T_f$ given from some node u to some node u', we use the following slot switching strategy to preserve Table condition 3.

Switching strategy: If x belongs to some group $g \in T_f$ in which no other slot is assigned to u', we are done. Otherwise, there must be a group g' with no slot assigned to u' since otherwise u' would have more than $\frac{s}{\alpha}$ slots at the end, violating Condition 2. Since Condition 3 was true before the movement, there must be a slot x' in g' that is assigned to some node u'' having no slot in g. Then, switch slots x and x' among u' and u'' which repairs Condition 3.

2. **Movement stage:** After the pairing stage, we only have nodes left that all want to decrease or increase their multiplicities. We consider these node by node. For each node v among these, we determine $\mu_f^{C'}(v)$, and then either move slots to v or away from v using the slot switching strategy in the pairing stage, if necessary, until v satisfies Condition 2.

Of course, it is not obvious that suitable slots can always be found for the reassignments (besides the pairing stage in which the nodes v and w still violate condition 2), but the following lemma implies that this is possible. In it, $C''(f)$ represents the intermediate multiset during the process of moving from $C(f)$ to $C'(f)$.

Lemma 3.51. *In any situation in which $|C''(f)|$ is within $|C(f)|$ and $|C'(f)|$, Conditions 1 and 3 are true, and at most one node violates Condition 2. But then, Condition 2 can be repaired for it so that all table conditions are met, w.h.p.*

Proof. For any node $w \in C''(f)$, let s_w be the number of slots w has in the table T_f. Let v be the node that is violating Condition 2. Then, v either needs additional slots or has to give up slots. Suppose first that v needs additional slots. In this case,

$$s_v \;<\; (1-\gamma)\,\frac{\mu_f^{C'}(v)}{\alpha}.$$

As long as there is a node w with

$$s_w - 1 \;\geq\; (1-\gamma)\,\frac{\mu_f^{C'}(w)}{\alpha},$$

we can move a slot from w to v until

$$s_v \;\geq\; (1-\gamma)\,\frac{\mu_f^{C'}(v)}{\alpha}.$$

Then, we repaired Condition 2 for v without violating the Condition 2 for any of the other nodes.

Suppose, however, that we reach a point in which $s_v < (1-\gamma)\,\mu_f^{C'}(v)/\alpha$ but there is no node w any more with $s_w - 1 \geq (1-\gamma)\,\mu_f^{C'}(w)/\alpha$. In this case, the total number of slots occupied by the nodes in $C''(f)$ is less than

$$\begin{aligned}(1-\gamma)\,\frac{\mu_f^{C'}(v)}{\alpha} + \sum_{w\neq v}\left[(1-\gamma)\,\frac{\mu_f^{C'}(w)}{\alpha}+1\right] \;&<\; \frac{1-\gamma}{\alpha}\sum_{w\in C''(f)}\mu_f^{C'}(w) + |C''(f)| \\ &=\; \left(\frac{1-\gamma}{\alpha}+1\right)|C''(f)|.\end{aligned}$$

If $\gamma \geq \frac{\beta}{1-\beta} + \alpha$, it follows from Lemma 3.46 that this is at most

$$\frac{1-2\beta}{\alpha(1-\beta)} \cdot |C''(f)| \;\leq\; \frac{1-2\beta}{\alpha(1-\beta)} \cdot (1+\beta)\, s \cdot r$$

w.h.p. Furthermore, it holds that

$$(1-2\beta)(1+\beta) \;=\; 1-\beta-2\beta^2 \;<\; 1-\beta,$$

thus, $\sum_w s_w < s \cdot \frac{r}{\alpha}$, which is a contradiction since all slots must be owned by a node at any time. Hence, it will always be possible to reassign slots so as to repair Condition 2 for node v in this case.

Next, we consider the case that node v needs to give up slots. In this case, v gets stuck if $s_v > (1+\gamma)\,\mu_f^{C'}(v)/\alpha$ and for all other nodes w, it holds that $s_w + 1 > (1+\gamma)\,\mu_f^{C'}(w)/\alpha$. Then, the total number of slots occupied by the nodes in $C''(f)$ is more than

$$\begin{aligned}
(1+\gamma)\,\frac{\mu_f^{C'}(v)}{\alpha} + \sum_{w \neq v} \left[(1+\gamma)\,\frac{\mu_f^{C'}(w)}{\alpha} - 1 \right] \;&>\; \frac{1+\gamma}{\alpha} \sum_{w \in C''(f)} \mu_f^{C'}(w) - |C''(f)| \\
&\geq\; \left(\frac{1}{\alpha(1-\beta)} + 1 \right) \sum_{w \in C''(f)} \mu_f^{C'}(w) - |C''(f)| \\
&\geq\; \frac{1}{\alpha(1-\beta)} \cdot (1-\beta)\, s \cdot r \;=\; s \cdot \frac{r}{\alpha}
\end{aligned}$$

w.h.p. This, however, is a contradiction. Hence, slots from node v can be reassigned to other nodes until Condition 2 holds for v. ■

Next, we bound the number of slot reassignments. A *step* in the movement or pairing stage is defined as the process of fixing the table conditions after the multiplicity of a node or pair of nodes has changed by 1.

Lemma 3.52. *In each step of the pairing or movement stage, at most* $2\frac{(1+\gamma)}{\alpha}$ *slots have to be reassigned in order to repair the table conditions.*

Proof. First, consider the movement stage. Suppose that the multiplicity of some node v increases by 1, that is, $\mu_f^{C'}(v) = \mu_f^{C} + 1$. We know that for its old multiplicity $\mu_f^{C}(v)$, it holds that

$$s_v \;\geq\; (1-\gamma)\,\frac{\mu_f^{C}(v)}{\alpha}.$$

Hence, at most $\frac{1}{\alpha}$ slots have to be moved to v to satisfy

$$s_v \;\geq\; (1-\gamma)\,\frac{(\mu_f^{C'}(v))}{\alpha}. \tag{3.17}$$

Since each slot movement may require a flip with another slot to repair Condition 3, the total number of slot reassignments is at most $\frac{2}{\alpha}$ in this case.

For the case that the multiplicity μ_f^C of some node v decreases by 1, at most $2\frac{(1+\gamma)}{\alpha}$ slots have to be reassigned. The worst case happens if $\mu_f^C(v)$ was previously 1 and v had $\frac{1+\gamma}{\alpha}$ slots.

Next, consider a step of the pairing stage. Suppose that for node v the multiplicity $\mu_f^{C'}(v) = \mu_f^C(v) - 1$ whereas the multiplicity of node w is $\mu_f^{C'}(w) = \mu_f^C(w) + 1$. Then, v has to give up at most $\frac{1+\gamma}{\alpha}$ slots while w has to get at most $\frac{1}{\alpha}$ slots in order to repair Condition 2. In any step of repairing Condition 2 for v and/or w (v gives a slot to w, or v gives up a slot, or w gains a slot), at most 2 slot reassignments are necessary, so altogether, at most $2\frac{(1+\gamma)}{\alpha}$ slot reassignments are needed to repair Condition 2 for the nodes v and w. ■

Hence, given a total change in the multiplicities of the nodes by μ, at most $2\mu\frac{(1+\gamma)}{\alpha}$ slot reassignments are necessary to get from $C(f)$ to $C'(f)$. Given k slot reassignments, the probability that a specific copy of a data item $d \in U$ with $g_1(d) \in f$ needs to be replaced is equal to

$$\frac{k}{r \cdot \frac{s}{\alpha}}.$$

Thus, the expected number of copy movements is at most

$$\frac{2\mu\frac{(1+\gamma)}{\alpha}}{r \cdot \frac{s}{\alpha}} \cdot |f| \cdot r|D| = \frac{2(1+\gamma)\mu|f|}{s} \cdot |D|$$

where $D \subseteq U$ is the set of data items that are stored in the system. A change in multiplicities by μ can be charged to a capacity change of $c(f) \geq \frac{|f|\mu}{s \cdot r}$ with respect to f in the past, and the way we perform interval rounding makes it possible that every capacity change is charged at most once. Thus, with respect to $c(f)$, the expected number of copy movements is at most

$$\frac{2(1+\gamma)s \cdot r \cdot c(f)}{s} \cdot |D| = 2(1+\gamma)c(f) \cdot r|D|$$

According to Fact 3.41, a capacity change of $c(f)$ requires the replacement of at least $c(f) \cdot r\frac{|D|}{2}$ copies for the copy distribution to remain fair with respect to f. Hence, for clean subframes, SPREAD is amortized $4(1+\gamma)$-adaptive.

Updating a dirty subframe f. If f is dirty, we maintain two tables for f. One table, T_1, for the interval f_1 from the starting point of f till the endpoint of $I(v)$, and one table, T_2, for the interval f_2 from the endpoint of $I(v)$ till the endpoint of f. The multisets $C(f_1)$ and $C(f_2)$ are equal to $C(f)$ with the difference that $C(f_1)$ contains a copy of node v while

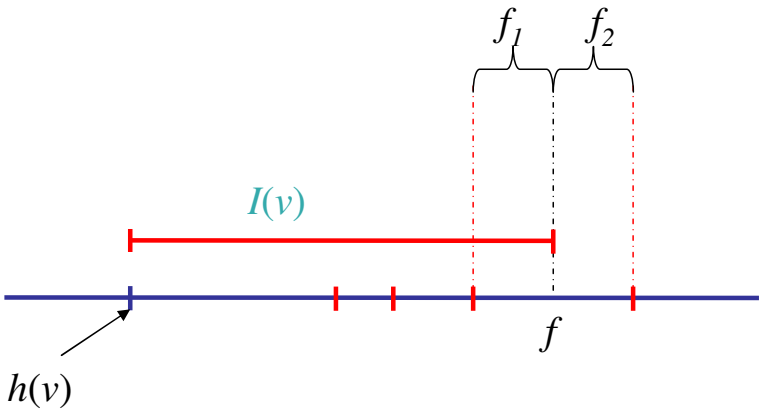

Figure 3.10: A Dirty Subframe

$C(f_2)$ does not contain v. Our goal is to make sure that T_1 and T_2 differ in at most $2\frac{(1+\gamma)}{\alpha}$ slots, which we call the *proximity condition*. This is ensured by the following strategy.

Suppose that $C(f)$ stays the same but the size of $I(v)$ changes. Then, we only update f_1 and f_2 accordingly and leave the tables T_1 and T_2 as before, which satisfies the proximity condition. If $C(f)$ only changes because the endpoint of $I(v)$ enters f from below, then T_2 inherits the table of f, and T_1 is obtained by applying slot reassignments for the node v to table T_2 until the table conditions are met for f_1. According to Lemma 3.52, this requires the reassignment of at most $2\frac{(1+\gamma)}{\alpha}$ slots, so the proximity condition holds afterwards. If $C(f)$ only changes because the endpoint of $I(v)$ is moving below f, then T_2 is chosen as the table for f. Similar solutions can be found if the endpoint of $I(v)$ enters or leaves f from above.

In all other cases, we first ignore a potential change in $I(v)$, which means that the sizes of f_1 and f_2 remain the same. We then adapt the table T_i of the interval f_i of largest size among f_1 and f_2 as described for the table of a clean subframe f to get from $C(f)$ to $C'(f)$ (ignoring changes in $C(f)$ due to $I(v)$ entering or leaving f), and then we construct the table of the other interval by performing at most $2\frac{(1+\gamma)}{\alpha}$ further slot reassignments in order to remove or add slots for node v. Afterwards, we update the sizes of f_1 and f_2 if necessary (i.e. if $I(v)$ has changed).

Next, we bound the adaptivity of SPREAD for dirty subframes. Suppose that $C(f)$ stays the same but the size of $I(v)$ changes by some ℓ in f (i.e. $\ell|f|$). Then, this can be charged to a change in capacity of node v of $c = \frac{\ell}{s \cdot r}$. Hence, the proximity condition ensures that the expected number of copy movements is at most

$$\frac{\ell \cdot 2\frac{(1+\gamma)}{\alpha}}{r \cdot \frac{s}{\alpha}} \cdot r|D| \;=\; 2(1+\gamma)c \cdot r|D|$$

which implies that, with respect to c, SPREAD is $4(1+\gamma)$-adaptive in this case.

If $C(f)$ only changes because the endpoint of $I(v)$ enters f from below or is moving below f, and this is associated with a change of $I(v)$ of length ℓ with respect to f, then it follows analogously to the first case that SPREAD is $4(1+\gamma)$-adaptive.

It remains to consider any remaining case. Let $\mu \geq 1$ be the total change in multiplicities in $C(f)$ (ignoring the change caused by $I(v)$). W.l.o.g., suppose that $|f_1| \geq |f_2|$. Then, at most

$$2\mu \frac{(1+\gamma)}{\alpha}$$

slots are reassigned in T_1 and at most

$$(\mu+2)\,2\,\frac{(1+\gamma)}{\alpha}$$

slots are reassigned in T_2. The latter bound holds because T_2 differs in at most $2\frac{(1+\gamma)}{\alpha}$ slots from T_1 before and after the reassignments.

Since a total change of μ can be charged to a capacity change of $c(f) \geq \frac{|f|\mu}{s\cdot r}$ with respect to f in the past, $|f_1| \geq |f_2|$, and $\mu \geq 1$, it follows from the arguments for clean subframes that the expected number of copy movements due to these changes is at most

$$\begin{aligned}\left(\frac{\mu\cdot 2\frac{(1+\gamma)}{\alpha}}{r\cdot\frac{s}{\alpha}} + \frac{(\mu+2)\,2\frac{(1+\gamma)}{\alpha}}{r\cdot\frac{s}{\alpha}}\right)\frac{|f|}{2}\cdot r|D| &\leq \frac{2\mu|f|\cdot 2\frac{(1+\gamma)}{\alpha}}{r\cdot\frac{s}{\alpha}}\cdot|D| \\ &\leq \frac{4(1+\gamma)s\cdot r\cdot c(f)}{s}\cdot|D| \\ &= 4(1+\gamma)\,c(f)\cdot r|D|\end{aligned}$$

For any additional changes due to $I(v)$, SPREAD is $4(1+\gamma)$-adaptive. Hence, overall SPREAD is amortized $8(1+\gamma)$-adaptive for dirty subframes. Summing up the adaptivity bounds over all subframes results in an amortized adaptivity of $8(1+\gamma)$.

3.5.2.5 Adaptivity.

In order to get from amortized adaptivity to adaptivity, we replace the deterministic rounding rules for the intervals above by a randomized rounding rule. More specifically, we choose an additional (pseudo-)random hash function $h' : V \to [0,1)$, and for every interval $I(w)$ and subframe $f = [x,y)$ in F_v with $v \neq w$ (or $|I(w)| \geq |F_w|$), we check the following:

Randomized rounding conditions:

1. Whenever the endpoint of $I(w)$ crosses $x + \frac{h'(v)}{|f|} \mod 1$ from below, $\mu_f^C(w)$ is increased by 1.
2. Whenever the endpoint of $I(w)$ crosses $x + \frac{h'(v)}{|f|} \mod 1$ from above, $\mu_f^C(w)$ is decreased by 1.

With this rule we obtain the following result:

Lemma 3.53. *For any capacity change, SPREAD is* $8(1+\gamma)$*-adaptive.*

Proof. First, suppose that n does not change. Consider any capacity change in the system, and for any node v, let $\Delta(I(v))$ be the interval representing the difference between $I(v)$ before and $I(v)$ after the change. Suppose that the starting point of $\Delta(I(v))$ is in some subframe f and the endpoint of $\Delta(I(v))$ is in some subframe f'.

First, consider the case that $\Delta(I(v)) \subseteq f$, i.e., $f = f'$. Then, it is easy to check that the probability that $\mu_f^C(v)$ increases or decreases by 1 is equal to $\frac{|\Delta(I(v))|}{|f|}$. We know from above that an increase or decrease of a multiplicity by 1 in a subframe f requires the replacement of an expected number of at most

$$\left(\frac{4(1+\gamma)|f|}{s}\right) \cdot |D|$$

copies in the system. Since c_v changed by $\Delta c_v = \frac{|\Delta I(v)|}{r \cdot s}$, this means that the expected number of copies replaced due to node v is at most

$$\left(\frac{|\Delta(I(v))|}{|f|}\right) \cdot \left(\frac{4(1+\gamma)|f|}{s}\right) \cdot |D| \;=\; 4(1+\gamma)\Delta c_v \cdot r\,|D|.$$

For $\Delta(I(v)) \not\subseteq f$, similar arguments also yield that the expected number of copies replaced due to v is at most $4(1+\gamma)\Delta c_v \cdot r|D|$.

Now, let us consider that n increases. Then, for any rounded $I(v)$ with endpoint in some subframe $f = [x,y)$ before the increase, $I(v)$ is only rounded again, applying the new decomposition, if $I(v) \subset I'(v)$ and the endpoint of $I(v)$ passes $x + \frac{h'(v)}{|f|}$ or y, or if $I'(v) \subset I(v)$ and the endpoint of $I(v)$ passes x or $x + \frac{h'(v)}{|f|}$, which preserves our adaptivity bound. ■

Combining all of the results in this section satisfies all given demands. Note that with the help of Chernoff bounds the adaptivity bound can also be shown to hold w.h.p. (up to minor order terms) if the total capacity change is $\omega(\varphi \log n)$, where φ is an upper bound on the maximum size of a subframe in the system. Hence, the smaller the subframes, the smaller will be the deviation from the expected number of replaced copies.

3.5.2.6 Variable number of copies per data item.

If the number of copies per data item varies but is upper bounded by r, then we just need to slightly adapt our storage strategy. For any data item with $r' \leq r$ copies, we first select a group of r distinct nodes as before and then store r' copies among r' of these nodes by selecting a (pseudo-)random starting node v in the group (via some additional hash

function) and then storing copies at the subsequent r' nodes in the group (where we treat the group as a ring). It is not difficult to show that this preserves all the properties shown above for data items with redundancy exactly r.

3.6 Conclusion and Open Problems

In this chapter, we observed and analyzed strategies for obtaining a redundant and fair distribution of normalized copies of data blocks among a given storage environment that may be considered static or, perhaps, dynamic. We showed, that when allocating the data items to disks of non-uniform capacity by some sequential balls-into-bins like algorithm, this will lead to an imbalance in the probability distributions of the allocations for each step $t \geq 2$. However, we presented an efficient data placement strategy, called COMB, that overcomes this imbalance problem such that for r identical copies of some data item, all copies are redundantly stored on r distinct disks of arbitrary capacity and furthermore, the fairness condition is satisfied. Unfortunately, since COMB is not adaptive and thus only works efficiently for static storage environments, after that, we introduced the adaptive SPREAD strategy which is efficient, always yields a redundant and fair data layout and is $O(1)$-adaptive for any change in the capacities of the disks in case of system scalings.

However, with respect to the observed allocation problem, the following open problems remain to be addressed in some future work.

- Determine (by some function) the shift in the assigned weights of the r-tuples when moving from a uniform to a non-uniform data placement.
- Investigate the given data redundant allocation problem (and thus, all presented algorithms) also for weighted data blocks.
- Find a multiple-choice approach for the standard balls-into-bins problem considering non-uniform capacities.
- Modify the r-tuple placement of the COMB strategy by the multiple-choice approach (and show (perhaps better) deviation bounds).
- Prove Conjecture 3.36.

CHAPTER 4

Erasure Codes for Reading and Writing

We now switch to allocation strategies that provide good results but only for classical storage environments (c.f. Section 1.1.2) and that use *erasure (resilient) encoding* to ensure fault-tolerance in a storage network. In general, erasure coding is a technique that has widely been employed for achieving high data availability and reliability in storage networks (e.g. [PGK88, PT03, Pla97, REG+03]) and communication systems (e.g. [ABB+97, NB97, BLM02, BLMR98, Riz97]).

From a general perspective, the main difference to replication-based strategies is that with erasure coding, the original information is no longer stored separately from the redundant data (copies) but is mixed up by the encoding yielding a codeword that contains the desired degree of redundancy. More specific, an erasure (resilient) code maps a word x of k symbols drawn from some finite set Σ, which is referred to as the *code alphabet*, into a *codeword* y of $n > k$ symbols from the same alphabet; the n symbols of y are stored separately on n distinct disks in the storage network. If then some disks fail for reading, in the optimal case, any k symbols from y are sufficient to recover x. That is, such codes can tolerate up to $n-k$ erasures which may be caused by failed, respectively temporarily not accessible disks (throughout this chapter, we refer to x as an *information word* or *message* of fixed size k, according to the *erasure channel model* introduced by Elias [Eli55]).

Although various erasure codes exist, the special set of codes we consider in the following are those for *RAID-like storage systems* such as RAID-arrays and SANs (e.g. [PGK88, PT03, Pla97, REG+03]) and whose most prominent representatives are parity codes like RAID and EVENODD [PGK88, BBBM94] or Reed-Solomon codes [HP03, Pla97] (we assume the reader to be somewhat familiar with basic properties of erasure codes in order to limit the scope of this thesis. However, a good overview can be found in e.g. [HP03, Rom96]). In contrast to other applications, RAID-like storage systems are much more sensitive concerning the size of the codeword because the separated storage of the code symbols induces additional storage costs. Hence, the codeword size has great influence on the required storage capacity but which is of less overhead than given with

replication, and this, clearly, is the very great advantage of erasure encoding in storage environments. Again, with replication, one has to cope with a storage overhead of a factor of $r-1$ that depends on the strict separation of the original and the redundant data (copies). Instead, with erasure coding, redundancy is directly encoded into the data symbols, and the ratio of original information to total code symbols is called the *(information) rate* of the code $r_c = \frac{k}{n} < 1$ which imposes a storage overhead (or stretch factor) $s_c = \frac{1}{r_c} = \frac{n}{k}$. Surely, it should hold that $s_c < 2k$ (otherwise, one would be better off using replication because of its simplicity). The least storage consumption of such codes is given by parity-based schemes that (considering $n = k$) require a factor $s_c = \frac{n+1}{n}$ (RAID) and $s_c = \frac{n+2}{n}$ (EVENODD). Additionally, since basing on simple XOR-operations, those schemes are highly suitable for hardware realization. Unfortunately, parity codes can merely tolerate a very small number of erasures at a time, what is often insufficient, especially in large-scale SANs. In that case, more complex codes, like Reed-Solomon codes, are applied that provide improved fault-tolerance.

However, compared to replication, the applied erasure codes lack of some substantial drawbacks, that are (a) to almost be applicable in uniform storage environments only, (b) to suffer from high complex algebraic operations for encoding/decoding, and (c) to lack of the ability to adapt to changing storage environments, but most importantly, all such codes suffer from a negative write and update behavior, respectively, because most of them strictly focus on providing an (almost) optimal data recovery. That is, for any k-sized information word x and corresponding codeword y, those codes aim at recovering the data from up to $n-k$ erasures (what is an optimal recovery behavior). Unfortunately, this optimal recovery property induces negative update behavior since, in particular, y is generated as a function of the original symbols implying that on every change of x, the complete codeword must also be updated, what is dismal. This overhead is widely known as *write-penalty*. Moreover, in a SAN, the write procedure is no longer atomic but rather describes a sequence of write operations which usually involves writing to multiple disks. These concurrent writes may then induce inconsistencies but which can, for instance, be reduced if only a small fraction of all parity symbols must be rewritten. Some approaches to reduce the update complexity are given by e.g. *X-codes* [XB99] but, however, such codes are limited to recover from any two simultaneous erasures only.

In the following, we overcome the update problem by the *Read-Write-Coding System (RWC)*, a very flexible class of erasure codes that are closely related to Reed-Solomon codes but offer enhanced flexibility and improved update behavior. However, before discussing the *RWC* system in more detail, we first require some brief introduction to linear block codes, the class of codes to which our Read-Write codes belong. To bound the scope of this thesis, we suppose the reader to be familiar with basics from linear algebra, number theory like e.g. modular arithmetic, and abstract algebra (e.g. group theory and the theory of finite fields) (see e.g. [HP03, Rom96, LN86] for overviews).

4.0.1 Preliminaries

Throughout this chapter, we address so called *linear block codes* which have useful properties for the storage systems we refer to. In particular, we consider static RAID-like storage systems that (a) are supposed to consist of a fixed sized set $[n]$ of disks for which furthermore a fixed allocation stripe size can be defined, and (b) for which, driven by mechanical components, access latency to the storage devices is magnitudes higher than the required time for encoding or decoding the data. With respect to the first issue, usually in those storage systems, so-called *block codes* are utilized in which all codewords have the same fixed length n and that have good potential to meet the desired disk utilization (in what follows, we state state some important facts about linear codes in very brief without proving the listed theorems; the proofs can be found in the references mentioned above).

Definition 4.1. (Block Code) *Let Σ be an arbitrary finite alphabet with $|\Sigma| \geq 2$. Then, an $[\mathbf{n},\mathbf{m}]$-**block code** over Σ is an m-elementary subset $C \subseteq \Sigma^n$ with $m \geq 1$. The elements of C are referred to as **codewords**, all of which have fixed length n, and the number $|C| = m$ of codewords in C is the size of the code.*

Now, to be more specific, the special class of block codes we concentrate on in what follows are *linear codes* which are defined by the means of linear algebra and thus, induce an easier description/analysis in general and with respect to the encoding/decoding operations. In particular, linear codes are codes over *finite fields*, that is, for a linear code, $\Sigma = \mathbb{F}_q$ and $\mathbb{F}_q$ is a finite field with q elements. Let $\mathbb{F}_q^k$ denote the k-dimensional vector space of all k-vectors over the finite field $\mathbb{F}_q$. Then, our k-sized messages are vectors $x = (x_1, x_2, \ldots, x_k) \in \mathbb{F}_q^k$, and we refer to $\mathbb{F}_q^k$ as the *message*, respectively *information space*. Furthermore, to introduce redundancy, we consider the following embedding of the vector space $\mathbb{F}_q^k$ into the n-dimensional vector space $\mathbb{F}_q^n$, $n > k$.

Definition 4.2. (Linear Code) *Consider the following injective linear mapping*

$$\gamma : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$$

*which defines a linear embedding of the message space $\mathbb{F}_q^k$ into $\mathbb{F}_q^n$. Then, the image $C = \gamma(\mathbb{F}_q^k)$ is a k-dimensional subspace of $\mathbb{F}_q^n$ that is isomorphic to $\mathbb{F}_q^k$, and to which we refer to as **linear** $(\mathbf{n},\mathbf{k})$**-code**, or simply $(\mathbf{n},\mathbf{k})$**-code**, of size $|C| = q^k$.*

It can be shown that every linear (n,k)-code C over the field $\mathbb{F}_q$ is an (n,q^k)-block code over the alphabet $\Sigma = \mathbb{F}_q$. The parameter n is called the *block length* and k is the *dimension* of the code.

The isomorphism between $\mathbb{F}_q^k$ and the (n,k)-code $C = \gamma(\mathbb{F}_q^k)$ is usually described by a $k \times n$ *generator matrix* Γ of rank k over the field $\mathbb{F}_q$ whose rows define a basis for C. Thus, we obtain

$$C = \gamma(\mathbb{F}_q^k) = \{v\Gamma \,|\, v \in \mathbb{F}_q^k\}.$$

For any set of k independent columns of Γ, the corresponding set of coordinates forms an *information set* for C. The remaining $r = n - k$ coordinates are termed a *redundancy set* and r is called the redundancy of C. If the first k coordinates form an information set, the code has a unique generator matrix of the form $(I_k \,|\, A)$ where I_k is the $k \times k$ identity matrix and we denote Γ to be in *standard form* as depicted in the following example.

Example 4.3. *The matrix* $\Gamma = (I_4 \,|\, A)$*, where*

$$\Gamma = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right),$$

is a generator matrix in standard form for a $(7,4)$ *binary code.*

The overall complexity of the encoding $v \to v\Gamma$ is given by the complexity of the multiplication $v\Gamma$ which is $O(n \cdot k)$ assuming $O(1)$-time for addition/multiplication in $\mathbb{F}_q$.

We now introduce a useful metric on $\mathbb{F}_q^n$ which, besides the block length and the dimension, is the third important parameter to determine the performance of a code.

Definition 4.4. (Hamming Metric) *Consider the function* $d : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{N}$ *defined as*

$$d(w,w') := |\{i \,|\, w_i \neq w'_i\}| \geq 0$$

and which satisfies the following properties:

1. *(non-negativity)* $d(w,w') \geq 0$ *for all* $w,w' \in \mathbb{F}_q^n$*,*
2. $d(w,w') = 0$ *if and only if* $w = w'$*,*
3. *(symmetry)* $d(w,w') = d(w',w)$ *for all* $w,w' \in \mathbb{F}_q^n$*,*
4. *(triangle inequality)* $d(w,w'') \leq d(w,w') + d(w',w'')$ *for all* $w,w',w'' \in \mathbb{F}_q^n$*.*

Then, d defines the so-called ***Hamming metric*** *on* $\mathbb{F}_q^n$*, and* $d(w,w')$ *is called the* ***Hamming distance*** *between the vectors* $w,w' \in \mathbb{F}_q^n$*. The pair* $(\mathbb{F}_q^n, d)$ *is referred to as a metric space, called the* ***Hamming space*** *of dimension n over the alphabet* $\mathbb{F}_q$*.*

Given the Hamming distance, we can now define the *minimum distance* of a code C which measures the difference between most similar codewords and that is an important invariant determining the error-correcting capability of C.

Definition 4.5. (Minimum Distance) *Let* C *be an* (n,k)*-code with* $|C| \geq 2$*. Then, the* ***minimum distance*** $d(C)$ *between distinct codewords is*

$$d(C)) := \min\{d(y,y') \,|\, y,y' \in C, y \neq y'\}.$$

Since $y \neq y'$, it follows that $d(C) \geq 1$. If the minimum distance of an (n,k)-code is known, we let $d := d(C)$ and refer to the code as an $(\mathbf{n},\mathbf{k},\mathbf{d})$**-code**. Then, we can state the following important theorem on the erasure correction capability of a linear code.

Theorem 4.6. *With an (n,k,d)-block code C, it is guaranteed that, after sending any codeword $y_i \in C$ over some noisy channel, y_i can completely be corrected if the number of erasures in y_i is at most $d-1$.*

Proof. Suppose by the way of contradiction that, after sending y_i, a word w is received containing $e \leq d-1$ erasures. Then, the pattern of $n-e$ correct symbols in w matches y_i, thus, the Hamming distance $d(w,y_i) \leq d-1$. If then there is another codeword y_j with $d(w,y_j) \leq d-1$, it follows that $d(y_i,y_j) \leq d-1$ because y_i and y_j can only differ in the $e \leq d-1$ coordinates. But this is a contradiction since according to Definition 4.5, there are no two codewords y_i, y_j with $d(y_i,y_j) \leq d-1$. ■

Therefore, the higher the minimum distance d of some (n,k,d)-code, the more erasures the code can tolerate. With respect to the block length, we call the ratio $\delta(C) = \frac{d}{n}$ the *relative distance* of the code which is a measure of the error-correcting capability of the code relative to its length. This ratio holds for any code (linear or not) of length n that has minimum distance d. Now, recall the *information rate* $r_c = \frac{k}{n} < 1$ as given in the introduction of this chapter. It holds that the higher the rate the higher the proportion of coordinates in a codeword actually containing information rather than redundancy, what makes the code less fault-tolerant. On the other hand, a high rate implies low storage overhead. As we can see, there is a trade-off between the rate and the relative distance when applying linear block codes in storage environments, and the major goal in coding theory is to find codes that have high rate as well as high relative distance (that is, both parameters should be close to 1). With respect to this objective, the following theorem states an important upper bound on the parameters (n,k,d).

Theorem 4.7. (Singleton Bound) *For every linear (n,k,d)-code over $\mathbb{F}_q$, it holds that*

$$d \leq n-k+1.$$

Therefore, according to the Singleton Bound, optimal codes are those for which equality $d = n-k+1$ holds. Such codes are called *maximum distance separable codes* (MDS codes) which can be defined as follows.

Definition 4.8. (MDS Code) *A linear (n,k)-code C is an* **MDS-*Code*** *if in each generator matrix Γ for C, any k columns are linearly independent.*

Corollary 4.9. *A linear (n,k)-code C is an* **MDS-*Code*** *if and only if any k components define an information set.*

It holds that no code of length n and minimum distance d has more codewords than an MDS code with parameters n and d. Trivial MDS codes, for instance, are the $(n, n-1, 2)$-parity codes in which a parity symbol is added to an information word of length $k = n$ representing the parity of the sum of all n information symbols as well as codes with parameters $(n, 1, n), (n, n, 1)$ who exist over every field $\mathbb{F}_q$ (c.f. Section 4.6).

Definition 4.8 already emphasized that the recovery capability of a linear code depends on the invertibility of the generator matrix Γ, thus, the ability to recover the information word from the codeword always requires an invertible $k \times k$ sub-matrix of Γ. In the following definition, we introduce the prominent *Vandermonde matrix* in which any $s \times s$ sub-matrix is invertible for some given s. Hence, the Vandermonde matrix has very useful properties for data recovery in linear block codes.

Definition 4.10. (Vandermonde Matrix) *Let $\alpha_1, \ldots, \alpha_s$ be elements in a field $\mathbb{F}$. The $s \times s$ matrix $V = [v_{i,j}]$, where*

$$v_{i,j} = \alpha_i^{j-1} = \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^{s-1} \\ 1 & \alpha_2 & \alpha_2^2 & \ldots & \alpha_2^{s-1} \\ 1 & \alpha_3 & \alpha_3^2 & \ldots & \alpha_3^{s-1} \\ 1 & \vdots & & \ddots & \vdots \\ 1 & \alpha_s & \alpha_s^2 & \ldots & \alpha_s^{s-1} \end{pmatrix},$$

*is called a **Vandermonde matrix**.*

Lemma 4.11. *The determinant* $\det V = \prod_{1 \le i < j \le s} (\alpha_j - \alpha_i)$. *In particular,* $\det V \neq 0$ *if the elements $\alpha_1, \ldots, \alpha_s$ are distinct.*

That is, V is nonsingular, and therefore, it is invertible.

4.0.2 Data Reconstruction from Failures

Besides providing an efficient computation of the algebraic operations used for encoding, the major goal when designing erasure codes for storage-based application is to ensure a more or less optimal data reconstruction in case of occurring erasures. In todays RAID-like storage systems, whenever obtaining high fault-tolerance is the major design focus (rather than efficient computation as given with parity codes), advanced MDS codes, like e.g. Reed-Solomon codes, are almost applied which can reconstruct the information from up to $n-k$ erasures as described in the following.

Consider the information vector $x = (x_1, x_2, \ldots, x_k)$, and let Γ be an $n \times k$ generator matrix. Then, a linear (n,k)-code C is represented by the following equation:

$$\Gamma x = y.$$

Furthermore, we can define functions $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ where each Γ_i denotes a linear combination of the information word that works on a word-by-word basis for generating the code symbol y_i:

$$y_i = \Gamma_i(x_1, x_2, \ldots, x_k) = \sum_{j=1}^{k} x_j \gamma_{i,j}$$

In other words, the functions Γ_i are the rows of Γ from which by multiplication with the vector x the complete codeword y is obtained. If we furthermore define Γ to be the $n \times k$ Vandermonde matrix, that is, $\gamma_{i,j} = \alpha_j^{i-1}$ (Definition 4.10) with pairwise distinct elements $\alpha_1, \ldots, \alpha_k \in \mathbb{F}_q$, the equation $\Gamma x = y$ becomes:

$$\begin{pmatrix} \gamma_{1,1} & \gamma_{1,2} & \cdots & \gamma_{1,k} \\ \gamma_{2,1} & \gamma_{2,2} & \cdots & \gamma_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{n,1} & \gamma_{n,2} & \cdots & \gamma_{n,k} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix} =$$

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \ldots & \alpha_k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_2^{n-1} & \alpha_3^{n-1} & \ldots & \alpha_k^{n-1} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Assuming that k components of y are available at the receiver, the information vector can be reconstructed by using any k equations that correspond to the known components of y and which form an invertible $k \times k$ sub-matrix Γ'. Surely, the extracted matrix Γ' is only invertible if the k chosen equations are linearly independent (but this is always the fact with a Vandermonde matrix).

Now, we describe a useful modification of the encoding that simplifies the data reconstruction in case one excepts rather few erasures. For this, we consider a code whose codewords y include a verbatim copy of the information blocks, and which we call *systematic*. This corresponds to including the identity matrix I_k in Γ as already shown in Example 4.3. We obtain the matrix $\Gamma_S = (I_k \,|\, \Gamma)$ and a codevector $y_S = (x \,|\, y)$ leading to the following equation ($\Gamma_S x = y_S$):

$$\begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ 1 & 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \ldots & \alpha_k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_2^{n-1} & \alpha_3^{n-1} & \ldots & \alpha_k^{n-1} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

The reconstruction of the information word x from any k available blocks of the codeword y_S is depicted in Figure 4.1.

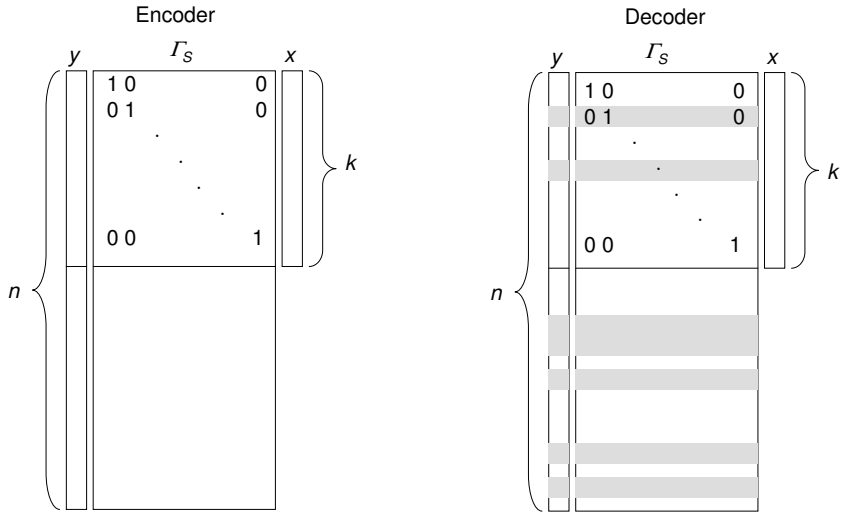

Figure 4.1: Encoding/Decoding in matrix form for a systematic code (the top k rows constitute the identity matrix I_k). y'_S and Γ'_S correspond to the grey areas of the vector and the matrix on the right hand side.

Reconstructing x from any combination of up to k symbols from y is always possible since Γ is a generator matrix and thus, any valid codeword y is a linear combination of its columns. In particular, as Γ has rank k, always any k rows are linearly independent imposing the linear mapping $x \to \Gamma x$ to be injective, thus, x can uniquely be determined from its corresponding codeword y. Moreover, the system $\Gamma' y' = x$ has a unique solution which can be obtained by Gaussian Elimination. Hence, x can be reconstructed from any subset of k code symbols because each code symbol conveys information on all the k source blocks (the ratio of this fraction of information is given by the rate).

However, any code that is able to reconstruct x from up to $n-k$ erasures suffers from a bad update behavior. In particular, if one information symbol changes from x_i to x'_i, any codesymbol y_i must also be modified. This can be effected by subtracting out the portion of the checksum word that corresponds to x_i, and adding the required amount for x'_i as follows:

$$y'_i = \Gamma_{i,j}(x_i, x'_i, y_i) = y_i + \gamma_{i,j}(y'_i - y_i).$$

Surely, both calculation and maintenance of the symbols y_i can be done by simple arithmetic but which are operations in the underlying finite field $\mathbb{F}_q$ which may be of different complexity depending on the field chosen. Furthermore, if any of the disks keeping y_i is not accessible, there is no chance to store the modified codeword (except merely the plain information word if a systematic code is applied such as given, for example, with the RAID 4/5 encoding).

Outline of this Chapter

In order to face the negative update behavior that is inherent to usual linear blocks codes and which turns to be pretty costly when such codes are applied in RAID-like storage environments, we introduce the *Read-Write-Coding-System (RWC)* – a very flexible framework for generating different linear block codes, called *Read-Write (RW) codes* in the following, which feature enhanced update properties for given codewords by simultaneously offering different degrees of fault-tolerance.

In particular, as we have seen, usual linear block codes merely consider a k-sized information vector x, a codeword y of fixed length n, and some linear function γ realizing the mapping from x to y. Instead, an RWC defines further parameters $k \leq r, w \leq n$ which offer enhanced possibilities to adjust the redundancy, and thus, the fault-tolerance capability of an RW code. In the language of coding theory, for any fixed r, an RWC provides linear (n,r,d)-codes over some finite field $\mathbb{F}_q$ that have (a) minimum distance $d = n - r + 1$ (thus, are MDS codes if $r = k$), and (b) any two codewords are within a distance of at most w from each other. More specific, an RWC generates appropriate subcodes of Reed-Solomon (RS) codes (see e.g. [HP03] for details on RS codes) of dimension r and length n in which every codeword has distance w. Depending on the values r, w and the field $\mathbb{F}_q$ chosen, different block codes can be generated, e.g. parity codes (if $q = 2$).

The ensured degree of redundancy mixed up with the improved update behavior offered by an Read-Write code provides significant benefits for the observed storage systems which, driven by the application's read and write behavior, on one hand, suffer from very costly I/O operations, and on the other hand, have to ensure some defined level of fault-tolerance at any time. Clearly, a Read-Write code provides best improvements for write-intensive applications because, given an n-sized codeword y and parameters r, w, it can decode the information from **any** r symbols of y whereas only **any** w symbols of y must be updated whenever the information word x changes completely (recall that we can choose $w < n$). This is different to other linear codes, like e.g. Reed-Solomon codes, which always must rewrite the codeword y completely as x changes.

An example. Consider a RAID 4-parity code with $n = 4$ hard disks storing a data file bit by bit, $\Sigma = \{0,1\}$. We encode $k = 3$ bits x_1, x_2, x_3 to symbols $y_1 = x_1, y_2 = x_2, y_3 = x_3$ and $y_4 = x_1 + x_2 + x_3$, where addition denotes the XOR-operation and the code symbols y_i, $1 \leq i \leq 4$, are stored separately on distinct disks. The XOR-operation enables to recover the original three bits from any combination of $r = 3$ hard disks, e.g. giving y_2, y_3, y_4 we have $x_1 = y_2 + y_3 + y_4$, $x_2 = y_2$, and $x_3 = y_3$. Thus, if any one disk is temporarily not available, reading data is still possible. However then, writing data is not possible since a complete change of the original information involves the change of the entire code; we call this *code consistency* (this also holds for any other erasure code applied in storage environments). The second example shows an RW-code. Again, consider $n = 4$

Contents		Code				Line
x_1	x_2	y_1	y_2	y_3	y_4	v
0	0	0	0	0	0	0
0	0	1	1	1	1	1
0	1	0	1	0	1	0
0	1	1	0	1	0	1
1	0	0	0	1	1	0
1	0	1	1	0	0	1
1	1	0	1	1	0	0
1	1	1	0	0	1	1

Table 4.1: A $(2,3,3,4)_2$-Read-Write-Code for contents x_1,x_2 and code y_1,y_2,y_3,y_4. Every information vector has two possible code words. Even if only three of the four code words are available for reading and writing, the system can perform read and write operations (see Figure 4.3 for the encoding).

hard disks with code bits y_1,y_2,y_3,y_4. Now, we encode $k = 2$ information bits x_1,x_2 such that any $r = 3$ code bits y_i,y_j,y_k can be used to recover the original message, and furthermore, only any of such $w = 3$ code bits $y_{i'},y_{j'},y_{k'}$ need to be changed to encode a completely new information. For instance, start with the codeword $(0,1,1,?)$. According to Table 4.1, the information is $(1,1)$ and therefore the complete code is $(0,1,1,0)$. Now, we want to encode $(0,1)$ without changing the second entry. For this, we choose line 0 for information $(0,1)$ and get code $(0,1,0,1)$.

Moreover, RW codes can exploit system information about existing erasures, which are caused by failed or blocked disks and that have rather long-term character in a SAN, for encoding and decoding. For instance, consider a codeword y with symbols stored on n disk from which $b \leq n-w$ disks are unreachable (e.g. failed or blocked). Then, using an RW code, y can still be updated to the codeword y' in a code consistent manner. Furthermore, if then some of the formerly blocked disks become available again while some other $b' \leq n-r$ disks turn to be unreachable, we can still recover the new information word x' by simply selecting any r of the remaining $n-b'$ accessible disks. Therefore, as long as sufficient disks are accessible, an RW code provides code consistent operations by circumventing blocked disks.

At last, some RW codes offer the possibility to change any of the parameters k,r,w,n during runtime, that is, a (k,r,w,n)-RW code can be changed to (nearly) any choice of (k',r',w',n') giving such codes the ability to adapt to changing system conditions.

From now on, we denote a $(k,r,w,n)_b$-Read-Write code a code which is given a k-symbol message x that is encoded into an n-symbol codeword y with symbols drawn from

some b-symbol alphabet Σ. Furthermore, we consider parameters r for recovering the information word and w for modifying a given codeword y, with $k \leq r, w \leq n$. In the next section, we state the operations of the RWC formally. After that, we prove general bounds for the parameters of RW codes and present a general scheme to generate $(k,r,w,n)_b$-RW codes as long as $k+n \leq r+w$ holds for an appropriate choice of b. Then, we introduce adaptive RW codes for which any of the given parameters can be subject to changes. At last, we investigate the question for which choices of (k,r,w,n) a coding system exists over the binary alphabet $\mathbb{F}_2 = \{0,1\}$ and discuss how RW codes can be combined. At last, notice that the following description is given in more operation-based terms rather than conceptual since RW codes base on the same well-studied algebraic principles as RS codes.

4.1 The Operational Model

Before we now introduce the concepts of the Read-Write codes in more detail, we first present the operations assigned to a Read-Write Coding System. Again, Read-Write codes encode information words into codewords. The information is given by a k-tuple over some finite alphabet Σ, and since Read-Write codes are linear block codes, the codeword is an n-tuple over the same alphabet, for $k < n$. For what follows, let $b = |\Sigma|$ and $\mathbf{P}(M)$ denotes the power set of some set M. Moreover, let $\mathbf{P}_\ell(M) := \{S \in \mathbf{P}(M) \mid |S| = \ell\}$.

Then, a $(k,r,w,n)_b$ Read-Write-Coding-System (RWC) provides the following operations.

1. **Initial state** $x_0 \in \Sigma^k$, $y_0 \in \Sigma^n$

   This is the initial state of the system with information x_0 and codeword y_0. This state is crucial because all further operations ensuring the beneficial features of an RW code depend on this initial state.

2. **Read function** $f : \mathbf{P}_r([n]) \times \Sigma^r \to \Sigma^k$

   This function reconstructs the information by reading r symbols of the codeword whose positions are known. The first parameter shows the positions (indices) of the symbols in the code, and the second parameter gives the corresponding code symbols. The outcome is the decoded information.

3a. **Write function** $g : \mathbf{P}_r([n]) \times \Sigma^r \times \Sigma^k \times \mathbf{P}_w([n]) \to \Sigma^w$

   This function adapts the codeword to a changed information by changing w symbols of the codeword at w given positions. The first two parameters describe the reading of the original information. Then, we have the new information as a parameter, and the last parameter indicates which code symbols to change in the codeword. The outcome are the values of the new w code symbols.

3b. **Differential write function** $\delta : \mathbf{P}_w([n]) \times \Sigma^k \to \Sigma^w$

This is a restricted alternative to the write function whose parameters are the positions S of symbols available for writing as well as the difference of the original information x and the new information x', but without reading the w code entries. The outcome is the difference of the available old and the new codeword symbols. Thus, for two functions $\Delta_1 : \Sigma^k \times \Sigma^k \to \Sigma^k$ and $\Delta_2 : \Sigma^w \times \Sigma^w \to \Sigma^w$ and w given positions from y, we can describe the write function g above by the differential write function as

$$y' = \Delta_2(y, \delta(S, \Delta_1(x, x'))).$$

All RW codes presented here have such differential write functions where Δ_1, Δ_2 denote the bit-wise XOR-operations. The goal is that e.g. a controller in a storage device i can, by itself, update its kept block y_i by simply adding (XOR) the received difference γ of y_i and y_i', i.e. $y_i' = y_i + \gamma$, if, for example, the device is blocked between reading the old and writing the modified block.

For a tuple $y = (y_1, \ldots, y_n)$ and a subset $S \in \mathbf{P}_\ell([n])$, let $\textsc{Choose}(S, y)$ be the tuple $(y_{i_1}, y_{i_2}, \ldots, y_{i_\ell})$ where $i_1, \ldots, i_\ell$ are the ordered elements of S. Furthermore, for an ℓ-tuple d, let $\textsc{Subst}(S, y, d)$ be the tuple where according to S each indexed element $y_{i_1}, y_{i_2}, \ldots, y_{i_\ell}$ of y is replaced by the element taken from d such that $\textsc{Choose}(S, \textsc{Subst}(S, y, d)) = d$ and all other elements in y remain unchanged in the outcome.
Now, for $S' \in \mathbf{P}_r([n])$, define the read operation

$$\mathbf{Read}(S', y) := f(S', \textsc{Choose}(S', y))$$

and for $S \in \mathbf{P}_w([n])$ and $x' \in \Sigma^k$, the write operation

$$\mathbf{Write}(S, S', y, x') := \textsc{Subst}(S, y, g(\mathbf{Read}(S', y), x', S)).$$

Since any Read-Write code needs to start at some initial state, we define the set of possible codewords Y as the *transitive closure* of the function $y \mapsto \mathbf{Write}(S, S', y, x')$ starting with $y = y_0$ and allowing all values S, S', x. Then, an RW code is correct if the following statements are fulfilled.

1. *Correctness of the initial state:* $\forall S \in \mathbf{P}_r([n])$:
   $\mathbf{Read}(S, y_0) = x_0$.

2. *Consistency of read operation*: $\forall S, S' \in \mathbf{P}_r([n])$
   $\forall y \in Y$: $\mathbf{Read}(S, y) = \mathbf{Read}(S', y)$.

3. *Correctness of write operation*:
   $\forall S \in \mathbf{P}_w([n]), \forall S' \in \mathbf{P}_r([n]), \forall y \in Y,$
   $\forall x \in \Sigma^n : \mathbf{Read}(S', \mathbf{Write}(S, S', y, x)) = x.$

Before we continue explaining the inherent mechanisms of an RWC in order to prove the operational statements above more formally, we first state some lower bounds on the parameters of an RWC that must be satisfied for the existence of RW codes at all.

4.2 Lower Bounds

The example of a $(2,3,3,4)_2$-RW code in the previous section stores two symbols of information in a four symbol code (c.f. Table 4.1). Unfortunately, this storage overhead of a factor two is unavoidable, as the following theorem shows (moreover, this implies that e.g. no $(3,3,3,4)_b$-RWC exists).

Theorem 4.12. *For parameters $r+w<k+n$ or $r,w<k$ and any base b, there does not exist any $(k,r,w,n)_b$-RWC.*

Proof. Consider a write operation and a subsequent read operation where the index set W of the write operation ($|W|=w$) and the index set R of the read operation ($|R|=r$) have an intersection: $W\cap R=S$ with $|S|=r+w-n$. Then, there are b^k possible change vectors with symbols in S that need to be encoded by the write operation since this is the only base of information for the subsequent read operation. This holds because all further $R\setminus S$ code symbols remain unchanged. Now, assume that $|S|<k$. Then, at most b^{k-1} possible changes can be encoded, and therefore, the read operation will produce faulty outputs for some write operations. Thus, $r+w-n\geq k$ and the claim follows.

If $r<k$, only b^r different messages can be distinguished while b^k different messages exist. Then, from the pigeonhole principle, it follows that such a code does not exist. For the case $w<k$, this is analogous. ■

From Theorem 4.12, it follows that in the best case $(k,r,w,n)_b$-RW codes have parameters $r+w=k+n$. We call such RW codes *perfect*. Unfortunately, such perfect codes do not always exist for any choice of parameters as the following lemma shows.

Lemma 4.13. *There is no $(1,2,2,3)_2$-RW code.*

Proof. Consider a read operation on the code bits y_1,y_2 and a write operation on y_2,y_3. Then, y_2 is the only intersecting bit which must be inverted in case of an information bit flip. The same holds for bit y_3 when considering a read operation on y_1,y_3 and a write operation on y_2,y_3. Thus, together, both y_2 and y_3 have to be inverted if the information bit x_1 flips. Now, consider a sequence of three write operations on bits $(1,2),(2,3),(1,3)$ each inverting the information bit x_1. After these operations, all code bits have been inverted twice bringing it back to the original state. In contrast, the information bit has been inverted thrice and is thus inverted. Therefore, all read operations lead to wrong results. ■

Contents	Code			Line
x	y_1	y_2	y_3	v
0	0	0	0	0
0	1	1	1	1
0	2	2	2	2
1	0	1	2	0
1	1	2	0	1
1	2	0	1	2
2	0	2	1	0
2	1	0	2	1
2	2	1	0	2

Table 4.2: A $(1,2,2,3)_3$-Read-Write code for an information word x and codeword y consisting of symbols y_1, y_2, y_3. For every information, there are three possible codewords, and if only any two of them three are available for reading and writing, the system can perform read and write operations (see Figure 4.4 in the next section for the encoding).

However, if we allow a larger symbol alphabet, we can find an RW code.

Lemma 4.14. *There exists a* $(1,2,2,3)_3$*-RW code.*

Proof. See Table 4.2 for an example. The correctness is straight-forward. ■

Clearly, concerning operational complexity, $b = 2$ (i.e. $\mathbb{F}_2$) is the best choice for codes applied in SANs because XOR-based I/O operations can often efficiently be realized in hardware. However, as common RAID 4/5 schemes as well as parity-based Reed-Solomon codes correspond to an $(n,n,n+1,n+1)_2$-RW code, $n \geq 1$, the following lemma shows that there is no parity-based placement scheme offering better update properties.

Lemma 4.15. *For* $n \geq 1$*, there is no* $(n,n,n,n+1)_2$*-RW code.*

Proof. The proof follows directly from Theorem 4.12. ■

4.3 Encoding/Decoding in the RWC

Now, we know that as long as the symbol alphabet is chosen sufficiently large, perfect RW codes always exist, and in the following, we show how perfect RW codes are generated by describing its encoding and decoding process in terms of vector arithmetic (surely, we always assume the symbol alphabet (i.e. the basis) to be sufficiently large whereas throughout this chapter, we use the terms basis and symbol alphabet interchangeably).

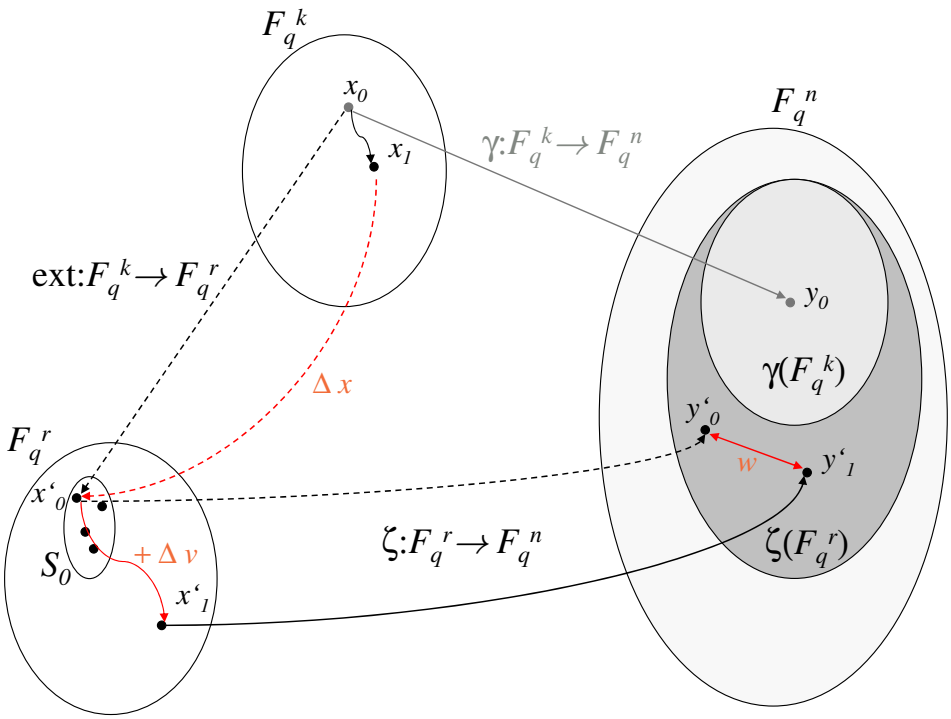

Figure 4.2: Illustration of Encoding/Decoding RW Codes

As already mentioned, Read-Write codes are closely related to Reed-Solomon codes and can thus be described by the same methods (i.e. by polynomials over finite fields). Nevertheless, in this thesis, we rather use a vector-based description that is better suited to sketch the encoding/decoding operations. In particular, for given information tuples $x = (x_1, \ldots, x_k) \in \Sigma^k$ whose base b is a sufficiently large finite field $\mathbb{F}_q$ and additional parameters $k \leq r, w \leq n$, we examine special subcodes of larger Reed-Solomon codes that have dimension r and length n and in which every codeword $y = (y_1, \ldots, y_n) \in \Sigma^n$ has distance at most w (Figure 4.2 depicts the generation of an RW code).

Every RW encoding starts with an initial information word x_0 generating an initial codeword y_0. As usual with linear codes, since we have chosen some finite field $\mathbb{F}_q$ for the basis, we can consider x_0 to be a vector in the vector space $\mathbb{F}_q^k$. In a usual linear encoding, x_0 would then be mapped directly into the vector space $\mathbb{F}_q^n$ of dimension $n > k$ by some linear function $\gamma : \mathbb{F}_q^k \to \mathbb{F}_q^n$ yielding y_0. More particularly, the resulting codeword y_0 lies in some k-dimensional subspace $\gamma(\mathbb{F}_q^k)$ of $\mathbb{F}_q^n$ that is isomorphic to the vector space $\mathbb{F}_q^k$, and as already illustrated in Section 4.0.2, if we suppose an MDS code, optimal recovery from up to $n-k$ erasures can then be achieved (recall that such codes have minimum distance $d = n-k+1$). However, as we also stated already, whenever the information word x_0 changes to x_1, all n code symbols of y_0 must be modified implying a distance $d = n$ between any two codewords in $\gamma(\mathbb{F}_q^n)$, what, again, is dismal in a SAN.

To improve the update behavior of linear codes such that, at most, $w < n$ symbols suffice to encode a complete information change, a Read-Write encoding takes the following detour concerning the vector mapping. First, depending on the parameter r, each information vector $x \in \mathbb{F}_q^k$ is extended to an r-sized vector x' by adding $\ell = r - k$ (slack) symbols $v'_1, \ldots, v'_\ell$ from the underlying symbol alphabet $\mathbb{F}_q$ that carry no particular information. In other words, we translate x into an r-dimensional vector $x' \in \mathbb{F}_q^r$ by some mapping function $ext : \mathbb{F}_q^k \to \mathbb{F}_q^r$. Importantly, (only) in case of the initial information word x_0, the ℓ symbols can be arbitrarily chosen from $\mathbb{F}_q$ yielding q^ℓ possible extension vectors x'_0 (denoted by the subset S_0 in Figure 4.2). Depending on the choice of the slack symbols, a vector $x'_0 \in S_0$ is fixed which is then mapped by some linear function $\zeta : \mathbb{F}_q^r \to \mathbb{F}_q^n$ to the codeword y'_0 lying in the subspace $\zeta(\mathbb{F}_q^r)$ of $\mathbb{F}_q^n$ (c.f. dashed lines). Clearly, $\zeta(\mathbb{F}_q^r)$ is an r-dimensional subspace of $\mathbb{F}_q^n$ which is isomorphic to $\mathbb{F}_q^r$. Moreover, as being also an (n,r)-linear MDS code, it is spanned by a set of n-dimensional basis vectors $b_1, \ldots, b_r$, and for given parameters n, k, r, a $(k,r,w,n)_b$-Read-Write code has minimum distance $d_{RW} = n - r + 1$ and is thus capable of tolerating up to $n - r$ erasures (see next section for a more formal description). Surely, this recovery property is not at good as offered by usual MDS codes but which is outweighed by the following improved update behavior.

Again, consider the initial information word x_0 and the corresponding RW codeword $y_0 \in \zeta(\mathbb{F}_q^r)$. Now, let x_0 somehow be changed/modified by the user/application what results in the new information word $x_1 = x_0 + \Delta x$ with Δx denoting the information difference between x_0 and x_1. Usually, in common linear (MDS) codes, this would create a new, independent encoding $\gamma(x_1) = y_1 \in \mathbb{F}_q^n$ of distance $d \in \{n - k + 1, \ldots, n\}$ whose upper bound cannot be improved. Instead, in a Read-Write code, in order to achieve that, at most, $w < n$ code symbols must be modified, any new encoding, i.e. any vector pair x'_i, y'_i, strongly depends on the initial setting x_0, y_0 and all subsequent vector pairs x'_j, y'_j for $0 < j < i$. The secret lies in appropriately adjusting the $\ell = n - w = r - k$ slack variables in each step. In particular, for an arbitrary step $i > 0$, consider the extended vector x'_i with slack variables v'_i valid for the corresponding codeword y'_i. Then, to achieve $y'_{i+1} = y'_i + \Delta y$ from $x'_{i+1} = x'_i + \Delta x$ in the following step, the new slack variables $v'_{i+1} = v'_i + \Delta v$ must be determined such that in the vector Δy exactly ℓ symbols remain unchanged (that is, are set to 0). From this, it follows that, at most, w symbols must be rewritten, and thus, the codeword space $\zeta(\mathbb{F}_q^r)$ corresponds to Y, the transitive closure of the initial codeword y_0 (c.f. Section 4.1).

4.3.1 The Matrix Approach

We now state the generation of RW codes formally for what we present a matrix-based approach which is similar to common linear encodings. In particular, we consider the information word $x = (x_1, \ldots, x_k) \in \mathbb{F}_q^k$, the corresponding codeword $y = (y_1, \ldots y_n) \in \mathbb{F}_q^n$,

and for any modification in x, let $\delta = \Delta x$ be the information change vector. Moreover, let $v = (v_1, \ldots, v_\ell)$ denote the vector of internal slack variables with $\ell = n - w = r - k$ and which carry no particular information. The linear mapping is given by an appropriate $n \times r$ generator matrix Γ with $\Gamma_{i,j} \in \mathbb{F}_q$; the sub-matrix $(\Gamma_{i,j})_{i\in[n],j\in\{k+1,\ldots,r\}}$ is called the *variable matrix*. Then, an RW code relies on the following equation realizing the mapping function $\zeta : \mathbb{F}_q^r \to \mathbb{F}_q^n$.

$$\begin{pmatrix} \Gamma_{1,1} & \Gamma_{1,2} & \cdots & \Gamma_{1,r} \\ \Gamma_{2,1} & \Gamma_{2,2} & \cdots & \Gamma_{2,r} \\ \vdots & \vdots & & \vdots \\ \Gamma_{n,1} & \Gamma_{n,2} & \cdots & \Gamma_{n,r} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_k \\ v_1 \\ \vdots \\ v_\ell \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{4.1}$$

Operations

- **Initialization**:

  Starting with an information vector $x_0 = (x_1, \ldots, x_k)$, the variables $(v_1, \ldots, v_\ell)$ can be set to arbitrary values (furthermore, if one wants to benefit from additional security features of this coding system (see Section 4.4), the slack variables must be chosen uniformly at random). The codeword $y_0 = (y_1, \ldots, y_n)$ is computed by Equation 4.1.

- **Read**: Given r code entries from y, compute x

  We rearrange the rows of Γ and the rows of y such that the first r entries of y are available for reading, and let y' and Γ' denote these rearranged vector and matrix, respectively. The first r rows of Γ' describe the $r \times r$ matrix Γ'' that we assume to be invertible. Then, the information vector x (and the variable vector v) is obtained by:

  $$\begin{pmatrix} x \\ v \end{pmatrix} = (\Gamma'')^{-1} y \; .$$

- **Differential write**: Given the information change vector δ and w code entries from y, compute the difference vector γ for the w code entries. Recall that y is updated by γ without first reading the information of y at the w code positions.

  The new information vector x' is given by $x'_i = x_i + \delta_i$. This notation allows to change the vector x' without reading its entries. Clearly, only the choices $w < r$ make sense. Now, according to Equation 4.1, given the new k-dimensional information vector x', the task is to find another $(r-k)$-dimensional vector ρ with $v' = v + \rho$ such that the new codeword

  $$y' = \Gamma \left(x' \mid v'\right)^T = \Gamma \left(x + \delta \mid v + \rho\right)^T$$

is a vector of distance at most w. Since we only consider at most w positions of y', we may, without loss of generality, assume that the last $n-w$ positions are zero such that $\Gamma\,(x' \mid v')^T = (y'_w \mid 0)^T$, with y'_w of length w, and the vector $\mathbf{0} = (0,\ldots,0)^T$ is of length $n-w$. Clearly, we must rearrange the rows of the matrix Γ due to the vector $(y'_w \mid 0)^T$. Thus, for simpler description, we partition Γ according to the lengths of the sub-vectors involved, that is, we rearrange the rows of Γ and y such that the writable code symbols are $y_1,\ldots,y_w$. Let Γ' denote this rearranged matrix and y' the rearranged code vector. Hence, we define the following sub-matrices of Γ'.

$$\begin{aligned}
\Gamma^{\leftarrow\uparrow} &= (\Gamma'_{i,j})_{i\in[w],j\in[k]},\\
\Gamma^{\uparrow\rightarrow} &= (\Gamma'_{i,j})_{i\in[w],j\in\{k+1,\ldots,r\}},\\
\Gamma^{\leftarrow\downarrow} &= (\Gamma'_{i,j})_{i\in\{w+1,\ldots,n\},j\in[k]},\\
\Gamma^{\downarrow\rightarrow} &= (\Gamma'_{i,j})_{i\in\{w+1,\ldots,n\},j\in\{k+1,\ldots,r\}}.
\end{aligned}$$

Given these definitions, we then obtain

$$\begin{aligned}
\left(\Gamma^{\leftarrow\uparrow}\right)x' + \left(\Gamma^{\uparrow\rightarrow}\right)v' &= y'_w\\
\left(\Gamma^{\leftarrow\downarrow}\right)x' + \left(\Gamma^{\downarrow\rightarrow}\right)v' &= 0.
\end{aligned}$$

Obviously, an important precondition of the write operation is the invertibility of the submatrix $\Gamma^{\downarrow\rightarrow}$. The code symbol vector is then updated by the w-dimensional vector

$$\gamma = ((\Gamma^{\leftarrow\uparrow}) - (\Gamma^{\uparrow\rightarrow})(\Gamma^{\downarrow\rightarrow})^{-1}(\Gamma^{\leftarrow\downarrow}))\;\delta,$$

such that the new codeword y' is derived from the former code symbols at the w given positions by simple addition, that is,

$$y' = y + \gamma.$$

In fact, the $(2,3,3,4)_2$-RWC in Table 4.1 can be generated by this matrix based approach (Figure 4.3 gives the encoding. Additionally, compare also the $(1,2,2,3)_3$-RWC in Table 4.2 and Figure 4.4).

The following definition and statements formalize the properties of the parameters of an RWC.

Definition 4.16. *An $n \times k$-matrix A over any base b with $n \geq k$ is **row-wise invertible** if each $k \times k$ matrix, constructed by combining k distinct rows of A, has full rank (and is therefore invertible).*

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ v_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

Readable code symbols	x_1	x_2
y_1, y_2, y_3	$y_1 + y_3$	$y_1 + y_2$
y_1, y_2, y_4	$y_2 + y_4$	$y_1 + y_2$
y_1, y_3, y_4	$y_1 + y_3$	$y_3 + y_4$
y_2, y_3, y_4	$y_2 + y_4$	$y_3 + y_4$

Write code	$(x'_1, x'_2) = (x_1 + \delta_1, x_2 + \delta_2)$			
	$y'_1 = y_1 +$	$y'_2 = y_2 +$	$y'_3 = y_3 +$	$y'_4 = y_4 +$
$1,2,3$	$\delta_1 + \delta_2$	δ_1	δ_2	0
$1,2,4$	δ_1	$\delta_1 + \delta_2$	0	δ_2
$1,3,4$	δ_2	0	$\delta_1 + \delta_2$	δ_1
$2,3,4$	0	δ_2	δ_1	$\delta_1 + \delta_2$

Figure 4.3: A $(2,3,3,4)_2$-Read-Write-Code over $\mathbb{F}_2 = \{0,1\}$ modulo 2.

Theorem 4.17. *The matrix-based RWC is correct and well-defined if the $n \times r$ generator matrix Γ as well as the $n \times (r-k)$ variable sub-matrix Γ' is row-wise invertible.*

Proof. Follows from the definition of row-wise invertibility and the description of the operations. To prove the correctness of the coding system, we show that after each operation Equation 4.1 is valid. This is straightforward for the initialization and read operations. It remains to prove the correctness of the write operation.

Again, consider the additive vector $(\rho_1, \ldots, \rho_\ell)$ denoting the changes in the variable vector v and the vector $(\gamma_1, \ldots, \gamma_w)$. With this and the information change vector δ, we obtain $x' = x + \delta$, $v' = v + \rho$ and $y' = y + \gamma$. The correctness of the write operation then follows by combining:

$$\Gamma \begin{pmatrix} x' \\ v' \end{pmatrix} = \Gamma \begin{pmatrix} x+\delta \\ v+\rho \end{pmatrix} = \Gamma \begin{pmatrix} x \\ v \end{pmatrix} + \Gamma \begin{pmatrix} \delta \\ \rho \end{pmatrix}$$

$$= \begin{pmatrix} y_1 \\ \vdots \\ y_w \\ y_{w+1} \\ \vdots \\ y_n \end{pmatrix} + \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_w \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

This equation is equivalent to the following.

$$\begin{aligned} (\Gamma^{\leftarrow\uparrow})\delta + (\Gamma^{\uparrow\rightarrow})\rho &= \gamma \\ (\Gamma^{\downarrow\rightarrow})\rho + (\Gamma^{\leftarrow\downarrow})\delta &= \mathbf{0}\,. \end{aligned}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Readable code symbols	x
y_1, y_2	$2y_1 + y_2$
y_1, y_3	$y_1 + 2y_3$
y_2, y_3	$2y_2 + y_3$

Writable symbols	$x' = x + \delta$		
	$y'_1 =$	$y'_2 =$	$y'_3 =$
y_1, y_2	$\delta + y_1$	$2\delta + y_2$	y_3
y_1, y_3	$2\delta + y_1$	y_2	$\delta + y_3$
y_2, y_3	y_1	$\delta + y_2$	$2\delta + y_3$

Figure 4.4: The $(1,2,2,3)_3$-Read-Write code over the alphabet $\mathbb{F}_3 = \{0,1,2\}$ modulo 3. corresponding to Table 4.2 above.

Since δ is given as a parameter, ρ can be computed as

$$\rho = \left(\Gamma^{\downarrow\rightarrow}\right)^{-1} \left(-\Gamma^{\leftarrow\downarrow}\right) \delta,$$

and γ by the last upper equation. If ρ is known, then the product $\Gamma \cdot (\delta \mid \rho)^T$ (reduced to the first w rows) gives the difference vector γ providing the new code entries of y' by $y' = y + \gamma$. ■

Theorem 4.18. *For parameters $k \leq r, w \leq n$ with $r + w = k + n$, there exists a $(k,r,w,n)_b$-RWC for an appropriate base b. Furthermore, this coding system can be computed in polynomial time.*

Proof. Follows from the following lemma and the fact that we use standard Gaussian elimination for recovery. ■

Lemma 4.19. *For each $n \geq k$ and basis $b \geq 2^{\lceil \log_2 n+1 \rceil}$, there is a row-wise invertible $n \times k$-matrix V over the finite field $\mathbb{F}_b$. Furthermore, each submatrix of V is also row-wise invertible.*

Proof. Define an $n \times k$ Vandermonde like matrix V consisting of non-zero and pairwise distinct elements $(\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_{[2^{\lceil \log_2 n+1 \rceil}]}$. Then, erase any $n-k$ rows what yields a $k \times k$ submatrix V' that is also a Vandermonde-matrix. Since all Vandermonde-matrices are invertible, the lemma follows. ■

4.4 Security and Redundancy

Induced by the usage of additional slack variables and besides adjustable fault-tolerance and improved update properties, Read-Write codes furthermore offer useful properties concerning data availability and security. Consider, for instance, the very extreme scenario of a combination of hard disks of n portable (laptop) computers within an office. If then a $(k,r,w,n)_b$-RWC is used for encoding of at most n laptops, it is sufficient if at least $\max\{r,w\}$ computers are accessible at the office at any time for data access and changes. If merely r computers are connected, at least the read operations can be performed. Now, what happens if computer hard disks are broken or information on some hard disks has changed ? Then, the inherent redundancy of the $(k,r,w,k)_b$-RWC allows to point out the number of wrong data and repair it (to some extent).

A different problem occurs if computers are stolen by some adversary to achieve knowledge about company data. The good news is that, for every matrix based RWC, it holds that one can give away any $n-w$ hard disks without revealing any information to the adversary. If the slack variables are chosen uniformly at random from Σ, the attacker will receive hard disks with perfect random sequences, absolutely useless without the other hard disks. As a surplus, this redundantizes the need for complex encryption algorithms.

Redundancy

Theorem 4.20. *Every $(k,r,w,n)_b$-RWC can detect and repair up to ℓ faulty code symbols if*

$$\frac{n!(r+\ell)!}{(n-\ell)r!} < \frac{1}{2}.$$

Additionally, it can reconstruct the data from $n-r$ missing code symbols (erasures).

Proof. The latter statement is trivial because if $n-r$ code symbols are missing, then by the definition of an RWC, the complete information can be recovered from any r accessible code symbols. Furthermore, if then ℓ out of these r code symbols are faulty, we can simply test every combination of the $\binom{n}{r}$ combinations of r received code symbols and take a majority vote over the information vector. In this vote, at least $\binom{n-\ell}{r}$ combinations produce the correct result what is a majority if

$$\binom{n-\ell}{r} \geq \frac{1}{2}\binom{n}{r}$$

which, by transformation, is equivalent to

$$\frac{n!(r+\ell)!}{(n-\ell)!\,r!} < \frac{1}{2}.$$

■

Security

If the coded symbols are stored on distinct storage devices, with an (k,r,w,n)-RWC, the loss of at most $n-\max\{r,w\}$ devices can be tolerated. For instance, if these storage devices were stolen, then the following theorem shows that the thief cannot reveal any information whatsoever from the encoded information. More particular, the attacker sees only a completely random sequence vector.

Theorem 4.21. *Every matrix based $(k,r,w,n)_b$-RWC for which $k+n=r+w$ holds can be used such that every choice of $n-w$ code symbols does not reveal any information about the original information vector.*

Proof. Choose random values for the slack variables $v_1,\ldots,v_\ell$ from the symbol alphabet Σ at initialization (let $|\Sigma|=b$). Then, there is an isomorphism between these slack variables and the stolen code symbols yielding a total of b^ℓ possibilities for the stolen code symbols to be changed. If more symbols are added, this starts to reveal some information. ■

4.5 Adaptive RW Codes

Usually in this chapter, we consider static scenarios for a SAN environment, but nevertheless, some perfect RW codes can be modified to also behave somehow adaptive in case of dynamic changes in the number of accessible disks (note that we still assume the storage devices to be of uniform size). More clearly, we already know from the previous chapter that in a SAN, adding and removing hard disks are the most delicate maneuvers. Provided that the size of the underlying symbol alphabet is chosen appropriately, we show in the following that perfect RW codes exist which allow to seamlessly continue all operations without forcing the system to be in some intermediate and, more importantly, invalid state.

Consider the following example. Assume 10 disk in a SAN using an $(8,9,9,10)$-RW code. Then, for better space utilization, the system administrator wants to switch to a $(4,7,7,10)$-RW code. In a usual encoding, this requires all disks to be available for the switch. Instead, in this section, we show that some special adaptive RW codes exist that allow to switch from one code to the other with only 9 disks being accessible for reading and writing. If then, after computing the re-encoding of all data on the 9 disks, the 10-th disk returns, it can immediately participate in the new $(4,7,7,10)$-RW code. Moreover, if the 10-th disk is permanently lost, it can be completely reconstructed from the new $(4,7,7,10)$-RW code.

In particular, an *adaptive RW code* is a set of certain RW codes $(k,r,w,n)_b$ with fixed alphabet and which, unlike the previously described codes, has a switch function. If a code is switchable, all parameters k,r,w,n can be subject to change. Furthermore, with respect to the codeword y, not all of the code symbols need to be read or changed.

Theorem 4.22. *Consider a sufficiently large constant M. Then, for a basis $b \geq 2^{\lceil \log_2 M+1 \rceil}$, there is an (M,b)-adaptive-RWC. In this system, it is possible to switch at any time from a $(k,r,w,n)_b$-RWC to any $(k',r',w',n')_b$-RWC provided that $n,n' \leq M$, and $k'+n' = r'+w'$ by merely reading any set of r encoded symbols and changing any set of w' encoded symbols.*

Proof. At first, we choose an appropriate base b by selecting a finite field $\mathbb{F}_q$ of sufficient size $q \geq 2^{\lceil \log_2 M+1 \rceil}$ and take the Vandermonde Matrix based approach as shown in Section 4.3.1. We change the main equation to the following.

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{M-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{M-1} \\ 1 & \vdots & & \ddots & \vdots \\ 1 & \alpha_{M-1} & \alpha_{M-1}^2 & \dots & \alpha_{M-1}^{M-1} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_k \\ v_1 \\ \vdots \\ v_{r-k} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \\ z_1 \\ \vdots \\ z_{M-n} \end{pmatrix}$$

Again, $x_1,\dots,x_k$ are the content symbols, $v_1,\cdots,v_{r-k}$ are the slack variables and $y_1,\dots y_n$ are the code symbols. The variables $z_1,\dots,z_{M-n}$ can be ignored for the beginning; they are neither contents, slack nor code symbols and can be generated from the content and slack symbols at any time. The initial vector as well as the read and write function are chosen as in the matrix based approach. Then, the switch operation, that is, switching from a (k,r,w,n)-RWC to a (k',r',w',n')-RWC, works as follows.

First, we read r code symbols at given positions and decode the vectors x and v according to the matrix based approach. Then, we adapt the size of the former code to the new code size. If $n' > n$, we compute the corresponding variables z_i from x and v. If $n' < n$, we rename $n-n'$ code variables to z-variables, and thus, reduce the code size. If $r' > r$, the content/slack-variable vector $(x \,|\, v)^T$ is extended by $(r'-r)$ 0-entries. We assume that new contents are written during the switch-operation (especially, if $k \neq k'$). For this, let $v'_1,\dots,v'_{r'-k'}$ be the new set of slack variables. Furthermore, we suppose at most w' code symbols (positions) available for writing.

We start by erasing the rows $n'+1,\dots,M$ in y and in the Vandermonde matrix since they are of no interest for this operation. Then, like in Section 4.3.1, we rearrange the residual matrix and the residual code vector such that the first w positions are the writable variables. We additionally rearrange the columns of the Vandermonde matrix and the contents/slack vector such that the new slack variables are on the rightmost columns, respectively lowermost lines. This results in the generator matrix Γ' (c.f. Section 4.3.1), and the original vector x is rearranged up to the lowest $r'-k'$ entries (possibly containing a mixture of old contents, old slack variables, and 0-entries). Let x' be the vector of the new

contents (adequately rearranged), and let v' be the new slack vector with $r'-k'$ entries. If $r' \geq r$, x has k' entries, and otherwise, x (x') has $r-r'$ additional entries resulting from former slack or content variables that must to be set to 0.

We first consider the case $r' \geq r$. The number of entries in x is k'. Then, we can perform an RWC write operation changing w' code symbols. Let $\ell' = r'-k' = n'-w'$ and partition Γ' like in Section 4.3.1. That is, let $\Gamma^{\leftarrow\uparrow}$ be a $w' \times k'$-sub-matrix of Γ', $\Gamma^{\uparrow\rightarrow}$ a $w' \times k'$-sub-matrix, $\Gamma^{\leftarrow\downarrow}$ an $\ell' \times n'$-sub-matrix and $\Gamma^{\downarrow\rightarrow}$ an invertible $\ell' \times \ell'$-sub-matrix of Γ'. Again, according to the matrix based approach, the new (rearranged) code vector y' is obtained by

$$y' = y + \left[(\Gamma^{\leftarrow\uparrow}) - (\Gamma^{\uparrow\rightarrow})(\Gamma^{\downarrow\rightarrow})^{-1}(\Gamma^{\leftarrow\downarrow})\right] \cdot (x'-x) \tag{4.2}$$

using the old (rearranged) writable symbol vector y. The proof of correctness is analogous to Section 4.3.1.

Now, consider the case $r' < r$. Then, the number of entries in x and x' is $\tilde{k} = k' + r - r'$. Again, let x' be the new (adequately rearranged) vector containing the $\tilde{k}$ new symbols, and v' is the new slack variable vector with $r'-k'$ entries. Note that $x'-x$ can be computed at this stage. We now perform a slightly adapted matrix based RWC write operation that changes w' code symbols. Clearly, compared to the previous case, that matrix consists of additional $r-r'$ columns but what does not cause any problem since we only have to adapt the sub-matrices. Furthermore, let $\ell' = r'-k' = n'-w'$ and $\tilde{w} = w+r-r'$. Then, let $\Gamma^{\leftarrow\uparrow}$ be a $\tilde{w} \times \tilde{k}$-sub-matrix of Γ', $\Gamma^{\uparrow\rightarrow}$ a $\tilde{w} \times \ell'$-sub-matrix, $\Gamma^{\leftarrow\downarrow}$ an $\ell' \times \tilde{k}$-sub-matrix and $\Gamma^{\downarrow\rightarrow}$ an invertible $\ell' \times \ell'$-sub-matrix of Γ'. As usual, y denotes the old and y' the new writeable symbols. Then, applying the defined matrices, the new vector y' is obtained as given with Equation 4.2, and again, the proof is analogous to the proof given in Section 4.3.1. ■

4.6 Boolean Read-Write-Codes

We already mentioned that the computational speed of RW codes strongly depends on the size of the finite field used (c.f. Reed-Solomon codes) because encoding/decoding is hardly driven by algebraic computations in the applied field, and the larger the field chosen the more complex the encoding/decoding process. Thus, the most interesting case for the choice of the alphabet is the binary case with $\Sigma = \{0,1\}$. Unfortunately, as we have shown in Lemma 4.13, there are no perfect Boolean $(1,2,2,3)$-RW codes, and furthermore, also for the matrix based RW codes (also called *parity Read-Write codes*, according to parity Reed-Solomon codes), the Boolean basis poses a severe restriction. The reason is that only for some dimensions Boolean matrices are row-wise invertible. We address these cases of valid $(k,r,w,n)_2$-RW codes in the following.

Lemma 4.23. *For each n, there exists an $n \times 1$ row-wise invertible Boolean matrix. Furthermore, for $k \geq 2$, there are $n \times k$ row-wise invertible Boolean matrices if and only if $n \in \{k, k+1\}$.*

Proof. The first claim is trivial. For the second, note there is a $(k+1) \times k$ row-wise invertible Boolean matrix, e.g. $\Gamma = (\Gamma_{i,j})_{i \in [k+1], j \in [k]}$ such that

$$\Gamma_{i,j} = \begin{cases} 1 & : \quad i = n \vee i = j \\ 0 & : \quad else \end{cases}$$

Now, we prove that there are no $(k+2) \times k$ row-wise invertible matrices as we show that, given a $k \times k$ full rank Boolean matrix Γ, there is always a unique vector leading to a $(k+1) \times k$ row-wise Boolean matrix.

Consider Γ as above and remove an arbitrary row i. Then, there are 2^{k-1} possibilities to add a row receiving a matrix with full rank. Such a row is described by the vector set $\Gamma(x_1, \ldots, x_k)^T$ where $x_i = 1$ and the rest is chosen arbitrarily. The only vector that can be added to each of these combinations is $\Gamma(1, \ldots, 1)^T$. After adding this vector, there is no other vector which is linearly independent from the other vectors. ■

This lemma has severe implications on the matrix based method for Boolean bases, as we show in the following.

Theorem 4.24. *If $r + w = k + n$, there are Boolean $n \times k$ generator matrices V for a valid $(k, r, w, n)_2$-RW encoding only for the following cases:*

1. $(1, 1, n, n)_2$, $(1, n, 1, n)_2$, $(k, k, k, k)_2$
2. *For $k \geq 2$:* $(k, k+1, k+1, k+2)_2$
3. *For $k \geq 1$:* $(k, k, k+1, k+1)_2$
4. *For $k \geq 1$:* $(k, k+1, k, k+1)_2$

Proof. Follows by combining the prerequisites of the matrix based RWC method with Lemma 4.23. ■

If we merely consider the encoding of a 1-bit information word, the following lemma holds.

Lemma 4.25. *Every $(1, r, w, r + w - 1)_2$-RW code is a matrix based RW code.*

Proof. Consider a write operation on the subset W of code symbols with $|W| = w$ and a read operation on the index set R with $|R| = r$ such that $|R \cap W| = 1$. Let i be the index of the element in the intersection. Now, if the information bit x flips to x', the code bit y_i must flip as well. Otherwise, the result of the read operation would be the same as before, what is incorrect. Thus, we can describe the operation on y_i by

$$y_i' \equiv y_i + x' + x \pmod 2.$$

Notice, there is always a set R with $|R| = r$ and $|R \cap W| = 1$ for any index $i \in W$. Thus, the congruence holds for all bits leading to the matrix representation of RW codes. ■

Unfortunately, there are no perfect Boolean RW codes of higher dimensions.

Lemma 4.26. *There is no $(2,r,w,n)_2$-RW code for $r+w = 2+n$ and $w \geq 4$.*

Proof. Consider arbitrary 4 bits of some codeword y and let their indices be given by the set $F = \{i,j,k,l\} \subseteq [n]$. Then, partition the residual index set $I_R = [n] \setminus F$ into two disjoint index sets R with $|R| = r-2$ and W with $|W| = w-4$. W.l.o.g. consider the set $F = \{1,2,3,4\}$ describing the code bits y_1, y_2, y_3, y_4. Now, consider write operations on the index sets $W_{i,j} = W \cup \{i,j\}$ for all distinct $i,j \in F$ and read operations on the index set $R_{i,j} = R \cup \{i,j\}$, again for all $i,j \in F$, imposing an intersection on at most two code symbols.

Now, what is the number of bits to flip whenever the information (x_1,x_2) changes to (x_1',x_2') (for our example, there are three change possibilities) ? To cover all cases, we define p_i as a predicate which is only true if y_i must be changed, i.e. inverted. If we then consider the read operation on the set $R_{i,j}$, clearly, all bits in R remain unchanged, and the only way the write operation induces a change for the read operation is that, at least, one of the code bits y_i, y_j is changed. To consider all such read operations, we have to fulfill the following term:

$$\bigwedge_{i,j \in [4], i \neq j} p_i \vee p_j =$$

$$(p_1 \wedge p_2 \wedge p_3) \vee (p_1 \wedge p_2 \wedge p_4) \vee (p_1 \wedge p_3 \wedge p_4) \vee (p_2 \wedge p_3 \wedge p_4).$$

As we see, all but (at most) one bit have to be inverted. Hence, there are five possibilities to encode all three possible changes to the information vector (x_1,x_2). Moreover, by the definition of the set $R_{i,j}$, each read operation can only observe two out of the four rewritten bits. In particular, the following cases occur (we start with $R_{1,2}$):

1. A: y_1 and y_2 are inverted.
2. B: Only y_1 is inverted.
3. C: Only y_2 is inverted.

Thus, A, B, and C can be mapped to all three possible changes of (x_1, x_2) to (x'_1, x'_2). Unfortunately, if we continue this game, we will observe a conflict with the read operation $R_{3,4}$ in the situations B and C that both hold for $R_{3,4}$, that is, both bits y_3 and y_4 must be inverted inducing an indistinguishable situation for the read operation. Thus, in other words, the read operation of $R_{3,4}$ cannot uniquely distinguish the change in the information vector. ■

This lemma can be generalized to the following theorem.

Theorem 4.27. *There is no $(k,r,w,n)_2$-RW code for $r+w = k+n$ and $w \geq k+2$.*

Proof. The proof is analogous to the proof of Lemma 4.26 but we now consider $k+2$ bits of y, and w.l.o.g. we set $F = [k+2]$ describing the code bits $y_1, y_2, \ldots, y_{k+2}$. Then again, we partition the $n-k$ residual code bits (i.e. the residual index set $I_R = [n] \setminus F$) into the sets R and W with $|R| = r-2$ and $|W| = w-k-2$. Let the considered read and write operations on the index sets $W_{i,j}$ and $R_{i,j}$ be defined as in Lemma 4.26.

Again, we are interested in the number of bits to flip if the information $(x_{i_1}, \ldots, x_{i_k})$ changes to $(x_{i_1}, \ldots, x_{i_{k+2}})$ (there are $2^k - 1$ change possibilities). Let the predicate p_i also be defined as before. If we then consider the read operation on $R \cup \{i_1, \ldots, i_k\}$, all bits in R remain the same. Thus, the only way the write operation induces a change is that, at least, one of the code bits $y_{i_1}, \ldots, y_{i_k}$ is changed (note that the read operation will read only two bits from this intersection). Considering all such read operations, we must now fulfill the following term:

$$\bigwedge_{S \in [k+2]:|S|=k} \bigvee_{i \in S} p_i \quad = \quad \bigvee_{S \in [k+2]:|S| \geq 3} \bigwedge_{i \in S} p_i.$$

Thus, at least 3 bits must be inverted in every write operation. Moreover, there are

$$2^{k+2} - \sum_{i=0}^{2} \binom{k+2}{i}$$

possibilities to encode all $2^k - 1$ possible changes to the vector $(x_1, \ldots, x_k)$.

However, in each read operation, only k bits are accessed (in the intersection) which requires $2^k - 1$ possible outcomes. If we then combine any two of the read operations, this implies for the whole set that, at least, $2^{k+2} - 7$ possible outcomes are demanded. Now, consider the intersection of read operations that have only $k-2$ positions in common. If the corresponding bits remain the same, clearly then, each of the residual pairs of two bits must differ, leaving 9 possibilities. In the other case, we have $2^{k-2} - 1$ possibilities for the common bit vector and 16 possibilities for the 'private' bit pairs. Hence, there are

$$\sum_{i=0}^{2} \binom{k+2}{i} - 7 > 1$$

codes missing, for all $k \geq 1$. ■

4.7 General RW-Codes

Since only few perfect Boolean RW codes exist, we now consider more general RW codes for which, in particular, hold

$$r+w > n+k.$$

In order to appropriately describe the existence of such codes, we use a modified matrix based approach in which all matrix entries are chosen uniformly at random from the alphabet Σ. Then, given this structures, we prove that with positive probability the read and write operations work.

We call the following approach a *random matrix* $(k,\ell,n)_b$*-RW code*. As usual, we consider the information vector x to consists of k symbols plus additional ℓ used slack variables, and an n-symbol code y generated by the matrix based encoding. Again, let $b = |\Sigma|$ denote the size of the symbol alphabet Σ. Furthermore, we consider an $n \times (k+\ell)$-generator matrix Γ whose elements are drawn (uniformly) at random from Σ. And as usual, we use the matrix based code representation

$$\Gamma(x \mid v)^T = y.$$

Then, the read and write operations can be derived as follows:

- **Read**

  Consider $r \geq k+\ell$ given readable positions in the code vector y. Then, we can derive the matrix Γ' by choosing all rows of Γ corresponding to the rows available for reading (c.f. Section 4.3.1). In Γ', we choose $\ell+k$ linear independent row vectors yielding matrix Γ'' (note that in case of an insufficient number of existing linear independent rows in the matrix (what may occur since the elements were chosen at random), the read operation fails). Corresponding to Γ'', we also reduce the code vector y yielding a codeword y' of size $k+\ell$ and compute the k-sized content vector x as well as the slack vector v as usual by

  $$(x \mid v)^T = (\Gamma'')^{-1} y'.$$

- **Write**

  Consider the $n \times l$ variable sub-matrix

  $$\Gamma^{\rightarrow} = (\Gamma_{i,j})_{i \in [n], j \in \{k+1,\ldots,k+l\}}.$$

There are $n-w$ non-writable positions corresponding to rows of the code vector y and the variable sub-matrix $\Gamma^{\rightarrow}$. In case that these $n-w$ rows of $\Gamma^{\rightarrow}$ are not linearly independent, clearly, the write operation fails (recall from Section 4.3.1 that a precondition for the write operation is the invertibility of the $(n-w)\times l$ variable submatrix $\Gamma^{\downarrow\rightarrow}$). Then, we could add some other $\ell-n+w$ linear independent rows of $\Gamma^{\rightarrow}$ (again, if these do not exist, the operation fails). Now, provided that we have found appropriate linearly independent rows, these rows then correspond to non-writable positions, and by using the complement of this rows as writable positions, we can directly apply the matrix based RW code write operation.

We now investigate the success probability of read and write.

Lemma 4.28. *Consider a $k\times(k+\ell)$ random matrix A over a given base $b\geq 2$. Then,*

$$\Pr[\,A \textit{ has an invertible } k\times k\textit{-sub matrix}\,] \;\geq\; 1-b^{-\ell+2}.$$

Proof. Since the column rank and row rank of a matrix are always equal, it is more convenient to argue the following by the column rank of the matrix A. Then, the probability that a random $k\times(k+\ell)$ submatrix of A does not have rank k equals the probability that all $k+\ell$ columns of A lie in some $k-1$-dimensional subspace of b^k. To sum up all these subspaces, we enumerate each subspace by its 1-dimensional orthogonal complement yielding a total of $\frac{b^k-1}{b-1}$ such subspaces (since we omit the zero vector, we have b^k-1 possibilities for a 1-dimensional vector or size k, and since for each such vector that spans some certain subspace, a multiplication with $b-1$ non-zero elements from the underlying base still spans the same subspace). Then, for each of such subspaces, the probability that all columns of the matrix are in this subspace is $b^{-(k+\ell)}$[1]. By the union bound, the probability that all the columns are in the same subspace is upper bounded by

$$b^{-(k+\ell)}\frac{b^k-1}{b-1} \leq \frac{b^{-\ell}}{b-1} \leq b^{-\ell}.$$

Thus, the probability that the matrix has full rank is at least $1-b^{-\ell}$. ■

From this, it follows that if only few combinations of read and write index sets occur, the overhead for general codes is small.

Theorem 4.29. *For given parameters r and w, consider a general RWC with read index sets $\mathscr{R}=\{R_1,\ldots,R_{|\mathscr{R}|}\}$, $|R_i|\geq r$, and write index sets $\mathscr{W}=\{W_1,\ldots,W_{|\mathscr{W}|}\}$, $|W_i|\geq w$. Then, there is a restricted $(k,r,w,n)_b$-RWC for any*

$$\begin{aligned} r &> k+\log_b|\mathscr{R}|+2,\\ w+r &> n+k+\log_b|\mathscr{W}|+2. \end{aligned}$$

[1] For each column vector c it holds: $\Pr[c \text{ lies in this subspace }] = b^{-1} = \frac{b^{k-1}}{b^k}$. In other words, one aims at sampling a vector from the k-dimensional space which is also in the $k-1$-dimensional subspace.

Proof. According to Lemma 4.28, the probability of having a valid code for reading by using r rows is at least $1 - b^{-(r-k)+2}$. Now, summing up the error probabilities of all possible read operations, we end up with

$$\sum_{i=1}^{|\mathscr{R}|} \Pr[\, R_i \text{ is not valid }] \quad \leq \quad |\mathscr{R}| b^{-(r-k)+2} \; < \; 1$$

provided that $r > k + \log_b |\mathscr{R}| + 3$.

For the write operation, we consider an $\ell \times (n-w)$ sub-matrix for $\ell = r-k$ in which we need to find $n - w \leq \ell$ independent rows, with positive probability. The other failure case can be omitted. Again, due to the previous lemma, with a success probability greater than $1 - b^{n-w-\ell+2}$ we succeed. Thus, for

$$n - w - r + k + 2 \quad > \quad \log_b |\mathscr{W}|,$$

the summed error probability is less than 1. Combining both error probabilities, the overall error probability is also less than 1. Hence, there exists a matrix allowing these restricted read and write index sets. ■

Yet, the overall number of possible read and write index sets is quite high. The following theorem shows that random matrices perform quite well for most of the read and write index sets.

Theorem 4.30. *For any base b, a random matrix based $(k,\ell,n)_b$-RW code successfully performs a read operation for a random choice of $k+\ell+q$ code symbols with probability $1 - b^{-q+2}$ and successfully performs a write operation for a random choice of $n-\ell+q$ writable code symbols with probability $1 - b^{-q+2}$.*

Proof. Follows directly from Lemma 4.28 and from the definition of the Read and Write operation of random matrix based $(k,\ell,n)_b$-RW-Codes. ■

4.8 Conclusion

In this chapter, we presented a useful extension of linear erasure codes, called Read-Write codes, for application in RAID-like storage systems. Such systems often store valuable business data implying some certain degree of reliability on the stored data. Furthermore, to exploit access parallelism, the data is distributed among n distinct storage devices but what makes the system subject to failures as the probability of failed disks increases with the number of devices connected. Therefore, fault-tolerant data placement schemes are highly required to beware from data loss. In addition, reducing system latency is also of great concern since access to the devices on which the data is stored is magnitudes slower than any memory access which leads to comparatively high I/O costs.

Usually, employed RAID schemes (RAID 4/5) and Reed-Solomon encoding are simple and efficient techniques that, on one hand, focus on aggregating disk performance by employing parallel access to them, and on the other hand, guarantee sufficient reliability to tolerate multiple disk failures, all at a low cost, but unfortunately, all such codes require n accessible disks for storing an n-block codeword appropriately, what leads to the following problems.

1. The modification of the original information implies access to all $n-k$ checksum disks; this write penalty leads to a performance bottleneck.

2. A complete modification of the information word results in inconsistencies if at least any of the n disks is missing.

3. Once the stripe size of an encoding is chosen, it cannot be changed any further, thus, it cannot be adjusted to potentially changing demands (e.g. changing degree of redundancy, adjusting to a higher number of tolerable failures, changing stripe size, changing document size, etc.).

RW codes can significantly reduce, or even eliminated these problems. In a perfect (k,r,w,n)-RW code (with $r+w=k+n$), any r disks suffice for reading, and any $\max\{r,w\}$ disks suffice for writing. The overhead is described by the stretch factor $\frac{m}{n}$, and any of these combinations (k,r,w,n) with $k \leq r,w \leq n$ can be chosen. In the case of adaptive RW codes, these parameters can be adjusted during runtime without reading and rewriting all disks. Furthermore, the information of $n-r$ disks can be recovered and the system can continue working if not more than $n-\max\{r,w\}$ disks fail. There is some redundancy against faulty hard disks. However, redundancy is better solved by reducing it with disk-wise checksums to the problem of failed disks. Interestingly, there is also a security feature which allows the theft of up to $n-w$ disks. The main advantage of RW codes in a SAN is that the current r, respectively w, fastest disks can be used for read or write operation. This leads to more efficient storage area networks.

In addition, in [DLS+04b], methods have been introduced for a wide-area distributed hash table (DHT) that provides high-throughput and low-latency network storage. The authors suggest a system called DHash++ which based on the peer-to-peer network Chord [SMK+01b], and one feature of DHash++ is to use an erasure-resilient code for storing data. In particular, they store each 8192-byte block as 14 1171-byte erasure-coded fragments, any seven of which are sufficient to reconstruct the block, using the IDA (Information Dispersal Algorithm) coding algorithm from [Rab89]. The benefit of this code is low latency as the fastest peers can be used to reconstruct the stored data. On the other hand, this system suffers from a bad update behavior making it a promising application for Read-Write-Codes to improve peer-to-peer-networks.

Bibliography

[ABB+97] Micah Adler, Yair Bartal, John W. Byers, Michael Luby, and Danny Raz. A modular analysis of network transmission protocols. In *Israel Symposium on Theory of Computing Systems*, pages 54–62, 1997.

[ABKU00] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 2000.

[ACMR95] Micah Adler, Soumen Chakrabarti, Michael Mitzenmacher, and Lars Rasmussen. Parallel randomized load balancing. pages 238–247, 1995.

[AT97] J. Alemany and J. S. Thathachar. Random striping news on demand servers. Technical Report TR-97-02-02, 1997.

[BBBM94] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: an optimal scheme for tolerating double disk failures in RAID architectures. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 245–254. IEEE Computer Society TCCA and ACM SIGARCH, April 18–21, 1994.

[BCSV00] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: the heavily loaded case. pages 745–754, 2000.

[BE07] André Brinkmann and Sascha Effert. Inter-node communication in peer-to-peer storage clusters. In *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies (MSST)*, San Diego, California, 24 - 26 September 2007.

[BEHV05] André Brinkmann, Sascha Effert, Michael Heidebuer, and Mario Vodisek. Distributed md. In *In Proceedings of the International Workshop on Storage Network Architecture and Parallel I/Os*, pages 81 – 88, Saint Louis, Missouri, USA, 18 September 2005.

[BEHV06] André Brinkmann, Sascha Effert, Michael Heidebuer, and Mario Vodisek. Realizing multilevel snapshots in dynamically changing virtualized storage environments. In *5th International Conference on Networking (ICN)*, number 5, Mauritius, 23 - 26 April 2006. Springer Verlag LNCS.

[BEMadHS07] André Brinkmann, Sascha Effert, Friedhelm Meyer auf der Heide, and Christian Scheideler. Dynamic and redundant data placement. In *27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007)*, Toronto, Canada, 25 - 29 June 2007.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[BGMJ94] Steven Berson, Shahram Ghandeharizadeh, Richard Muntz, and Xiangyu Ju. Staggered striping in multimedia information systems. pages 79–90, 1994.

[BHMadH+04] André Brinkmann, Michael Heidebuer, Friedhelm Meyer auf der Heide, Ulrich Rückert, Kay Salzwedel, and Mario Vodisek. V:drive - costs and benefits of an out-of-band storage virtualization system. In *Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST)*, pages 153 – 157, College Park, Maryland, USA, 13 - 16 April 2004.

[BLM02] J. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communications, 20(8), Oct*, 2002.

[BLMR98] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM'98*, pages 56–67, Sep 1998.

[BMadHS97] Petra Berenbrink, Friedhelm Meyer auf der Heide, and Klaus Schröder. Allocating weighted jobs in parallel. *SPAA 1997*, pages 302–310, June 1997.

[BMS+03] A. Brinkmann, F. Meyer auf der Heide, K. Salzwedel, C. Scheideler, M. Vodisek, and U. Rückert. Storage Management as Means to Cope with Exponential Information Growth. In *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medcine, and Mobile Technologies on the Internet, SSGRR 2003w*, January 2003.

[BSS00] A. Brinkmann, Kay Salzwedel, and C. Scheideler. Efficient, distributed data placement strategies for storage area networks. In *Proc. of the 12th ACM Symposium on Parallel Algorithms and Architectures (SPAA'00)*, pages 119–128, 2000.

[BSS02] A. Brinkmann, Kay Salzwedel, and C. Scheideler. Compact, adaptive placement schemes for non-uniform distribution requirements. In *Proc. of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA'02)*, pages 53–62, 2002.

[BSV03] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, February 2003.

[BSV04] André Brinkmann, Kay Salzwedel, and Mario Vodisek. A case for virtualized arrays of raid. In *Proceedings of the International Workshop on Storage Network Architecture and Parallel I/Os SNAPI 2004*, pages 9–16, Antibes Juan-les-pins, France, 30 September 2004.

[CL00] T. Cortes and J. Labarta. A case for heterogenenous disk arrays. In *Proc. of the IEEE International Conference on Cluster Computing (Cluster'2000)*, pages 319–325, 2000.

[CL01] T. Cortes and J. Labarta. Extending heterogeneity to RAID level 5. In *USENIX 2001*, Boston, June 2001.

[CLG+94] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, 1994.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[CW77] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, New York, NY, USA, 1977. ACM.

[DKM+88] Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert Endre Tarjan. Dynamic perfect hashing: Upper and lower bounds. In *IEEE Symposium on Foundations of Computer Science*, pages 524–531, 1988.

[DLS+04a] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a dht for low latency and high throughput. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 7–7, Berkeley, CA, USA, 2004. USENIX Association.

[DLS+04b] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a DHT for low latency and high throughput. In *NSDI*, pages 85–98, 2004.

[DR01] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. pages 75–80, 2001.

[ED88] R. J. Enbody and H. C. Du. Dynamic hashing schemes. *ACM Comput. Surv.*, 20(2):850–113, 1988.

[Eli55] P. Elias. Coding for two noisy channels. In *Information Theory, Third London Symposium*, pages 61–76. Butterswort's Scientific Publications, 1955.

[Fal07] FalconStor. *The Value of Storage Virtualization*. White Paper, FalconStor Software Inc., 2007.

[FNPS79] Ronald Fagin, Jürg Nievergelt, Nicholas Pippenger, and H. Raymond Strong. Extendible hashing - a fast access method for dynamic files. *ACM Trans. Database Syst.*, 4(3):315–344, 1979.

[Gan07] John F. Gantz et al. *ICD Study: The Expanding Digital Universe*. White Paper, EMC^2, March 2007.

[Gib99] Garth A Gibson. Redundant disk arrays: Reliable, parallel secondary storage. Technical report, Berkeley, CA, USA, 1999.

[GLS93] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, second corrected edition edition, 1993.

[GM00] Garth A. Gibson and Rodney Van Meter. Network attached storage architecture. *Commun. ACM*, 43(11):37–45, 2000.

[Gup02] M. Gupta. *Storage Area Network Fundamentals*. Cisco Press, 2002.

[HGK+94] Lisa Hellerstein, Garth A. Gibson, Richard M. Karp, Randy H. Katz, and David A. Patterson. Coding techniques for handling failures in large disk arrays. *Algorithmica*, 12(2/3):182–208, 1994.

[HJH02] Kai Hwang, Hai Jin, and Roy S.C. Ho. Orthogonal striping and mirroring in distributed raid for i/o-centric cluster computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):26–44, 2002.

[HM03] R. Honicky and E. Miller. A fast algorithm for online placement and reorganization of replicated data. 2003.

[HM04] R. J. Honicky and Ethan L. Miller. Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution. In *Proceedings of the 18th IPDPS Conference*, 2004.

[HP03] Cary Huffmann and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, Cambridge, UK, 2003.

[HR90] Torben Hagerup and C. Rüb. A guided tour of chernoff bounds. *Inf. Process. Lett.*, 33(6):305–308, 1990.

[Hro03] J. Hromkovic. *Algorithms for Hard Problems, 2nd Edition. (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[HZ05] J. Hromkovic and I. Zámecniková. *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[JK77] Norman L. Johnson and Samuel Kotz. *Urn models and their application*. John Wiley & Sons, New York-London-Sydney, 1977. An approach to modern discrete probability theory, Wiley Series in Probability and Mathematical Statistics.

[Joh84] Olin G. Johnson. Three-dimensional wave equation computations on vector computers. *Proceedings of the IEEE*, 72(1):90–95, January 1984.

[JT05] A. Telles J. Tate, R. Kanth. *Introduction to Storage Area Networks*. Technical report, IBM, May 2005.

[KLL+97] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[KLM92] Richard M. Karp, Michael Luby, and Friedhelm Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. pages 318–326, 1992.

[KLMadH96] Richard M. Karp, Michael Luby, and Friedhelm Meyer auf der Heide. Efficient pram simulation on a distributed memory machine. *Algorithmica*, 16(4/5):517–542, 1996.

[KSB+99] David Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, and Yoav Yerushalmi. Web caching with consistent hashing. In *WWW '99: Proceedings of the eighth international conference on World Wide Web*, pages 1203–1213, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[LMR98] W. Litwin, J. Menon, and T. Risch. LH* schemes with scalable availability. Technical Report RJ 10121 (91937), IBM Research, Almaden Center, May 1998.

[LN86] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, New York, NY, USA, 1986.

[LN96] Witold Litwin and Marie-Anne Neimat. High-availability LH* schemes with mirroring. In *Conference on Cooperative Information Systems*, pages 196–205, 1996.

[LNS93] Witold Litwin, Marie-Anne Neimat, and Donovan A. Schneider. Lh*: Linear hashing for distributed files. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 327–336, New York, NY, USA, 1993. ACM.

[LR02] Witold Litwin and Tore Risch. LH*$_G$: A high-availability scalable distributed data structure by record grouping. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):923–927, 2002.

[LS00] Witold Litwin and Thomas Schwarz. LH* RS : A high-availability scalable distributed data structure using reed solomon codes. In *SIGMOD Conference*, pages 237–248, 2000.

[LYD71] V. Y. Lum, P. S. T. Yuen, and M. Dodd. Key-to-address transform techniques: a fundamental performance study on large existing formatted files. *Commun. ACM*, 14(4):228–239, 1971.

[MadH06] Friedhelm Meyer auf der Heide. *Kommunikation in Parallelen Rechenmodellen*. Skript, University of Paderborn, July 2006.

[Mar79] G. N.N. Martin. Spiral storage: Incrementally augmentable hash addressed storage. Technical report, Coventry, UK, UK, 1979.

[Mas97] P. Massiglia. *RAB. The RAIDbook. A Storage System Technology Handbook.* RAID Advisory Board, 1997.

[McD89] C. McDiarmid. On the method of bounded differences. In J. Siemons, editor, *Surveys in Combinatorics*. London Mathematical Society Lecture Note Series 141, Cambridge University Press, 1989.

[MNR02] D. MALKHI, M. NAOR, and D. RATAJCZAK. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st annual ACM symposium on Principles of distributed computing*. ACM Press, 2002.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[MS08] Mario Mense and Christian Scheideler. Spread: An adaptive scheme for redundant and fair storage in dynamic heterogeneous storage systems. In *19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, San Francisco, California, USA, 20.-22. Febr.,, January 2008.

[Mul84] James K. Mullin. Unified dynamic hashing. In *VLDB '84: Proceedings of the 10th International Conference on Very Large Data Bases*, pages 473–480, San Francisco, CA, USA, 1984. Morgan Kaufmann Publishers Inc.

[NB97] J. Nonnenmacher and E. Biersack. Asynchronous multicast push: Amp. In *Proceedings of ICCC'97, pp. 419–430, Cannes, France, November*, 1997.

[Oto88] Ekow J. Otoo. Linearizing the directory growth in order preserving extendible hashing. In *Proceedings of the Fourth International Conference on Data Engineering*, pages 580–588, Washington, DC, USA, 1988. IEEE Computer Society.

[PGK88] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, pages 109–116, June 1988.

[Pla97] James S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software, Practice and Experience*, 27(9):995–1012, 1997.

[PS98] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Dover Publications, July 1998.

[PT03] J. S. Plank and M. G. Thomason. On the practical use of ldpc erasure codes for distributed storage applications. Technical Report CS-03-510, University of Tennessee, September 2003.

[Rab89] M. O. Rabin. Efficient dispersal of information for security, load balacing and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

[RD01] Antony I. T. Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Symposium on Operating Systems Principles*, pages 188–201, 2001.

[REG+03] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype. In *Proceedings of the Conference on File and Storage Technologies*. USENIX, 2003.

[Riz97] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2):24–36, April 1997.

[RL05] Rodrigo Rodrigues and Barbara Liskov. High availability in dhts: Erasure coding vs. replication. In Miguel Castro and Robbert van Renesse, editors, *IPTPS*, volume 3640 of *Lecture Notes in Computer Science*, pages 226–239. Springer, 2005.

[Roc70] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, New Jersey, 1970.

[Rom96] Steven Roman. *Introduction to coding and information theory*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[Ros00] K.H. Rosen. *Handbook of Discrete and Combinatorial Mathematics*. CRC; 1 edition, 2000.

[Ros06] Sheldon M. Ross. *Introduction to Probability Models, Ninth Edition*. Academic Press, Inc., Orlando, FL, USA, 2006.

[RS98] Martin Raab and Angelika Steger. "balls into bins" — A simple and tight analysis. *Lecture Notes in Computer Science*, 1518:159–??, 1998.

[Sal04] Kay Salzwedel. *Data Distribution Algorithms for Storage Networks*. Dissertation, Universität Paderborn, Heinz Nixdorf Institut, Theoretische Informatik, 2004. EUR 20,00 ISBN 3-935433-62-X.

[San01] P. Sanders. Reconciling simplicity and realism in parallel disk models. In *Proc. of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 67–76. SIAM, Philadelphia, PA, 2001.

[Sch00] C. Scheideler. *Probabilistic Methods for Coordination Problems*. HNI-Verlagsschriftenreihe 78, University of Paderborn, 2000.

[Sch01] Klaus Schröder. *Balls into Bins: A Paradigm for Job Allocation, Data Distribution Processes, and Routing*. Dissertation, Universität Paderborn, Heinz Nixdorf Institut, Theoretische Informatik, 2001. ISBN 3-931466-88-4.

[SM98a] J. R. Santos and R. Muntz. Performance analysis of the RIO multimedia storage system with heterogeneous disk configuration. In *ACM Multimedia 98*, pages 303–308, 1998.

[SM98b] Jose Renato Santos and Richard Muntz. Using heterogeneous disks on a multimedia storage system with random data allocation. Technical Report 980011, 18, 1998.

[SMK+01a] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the ACM SIGCOMM 2001*, pages 149–160, August 2001.

[SMK+01b] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[SMRN00] Jose Renato Santos, Richard R. Muntz, and Berthier A. Ribeiro-Neto. Comparing random data allocation and data striping in multimedia servers. In *Measurement and Modeling of Computer Systems*, pages 44–55, 2000.

[SS05] Christian Schindelhauer and Gunnar Schomaker. Weighted distributed hash tables. In *SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 218–227, New York, NY, USA, 2005. ACM Press.

[Tec07] Hitatchi Global Storage Technologies. *Hitatchi Ultrastar™A7K1000*. Specification Sheet, Hitatchi Global Storage Technologies, 2007.

[TPBG93] F. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming raid - a disk array management system for video files. In *Proceedings of of 1st ACM Multimedia*, pages 393–400, 1993.

[Voe99] Berthold Voecking. How asymmetry helps load balancing. In *IEEE Symposium on Foundations of Computer Science*, pages 131–141, 1999.

[VRG95] Harrick M. Vin, S. S. Rao, and Pawan Goyal. Optimizing the placement of multimedia objects on disk arrays. In *International Conference on Multimedia Computing and Systems*, pages 158–166, 1995.

[WAS+96] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal policies in network caches for World-Wide Web documents. In *Procedings of the ACM SIGCOMM '96 Conference*, Stanford University, CA, 1996.

[WBMM06] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. Crush: controlled, scalable, decentralized placement of replicated data. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 122, New York, NY, USA, 2006. ACM.

[Wie07] Udi Wieder. Balanced allocations with heterogenous bins. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 188–193, New York, NY, USA, 2007. ACM.

[WK02] Hakim Weatherspoon and John Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 328–338, London, UK, 2002. Springer-Verlag.

[XB99] Lihao Xu and Jehoshua Bruck. X-code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory*, 45(1):272–276, 1999.

[ZG97] Roger Zimmermann and Shahram Ghandeharizadeh. Continuous display using heterogeneous disk-subsystems. In *MULTIMEDIA '97: Proceedings of the fifth ACM international conference on Multimedia*, pages 227–238, New York, NY, USA, 1997. ACM.

[ZG00] Roger Zimmermann and Shahram Ghandeharizadeh. HERA: Heterogeneous extension of RAID. In *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, 2000.

[Zim98] Roger Zimmermann. *Continuous media placement and scheduling in heterogeneous disk storage systems*. PhD thesis, Los Angeles, CA, USA, 1998. Adviser-Shahram Ghandeharizadeh.