# SPREAD: An Adaptive Scheme for Redundant and Fair Storage in Dynamic Heterogeneous Storage Systems

Mario Mense[*] Christian Scheideler[†]

**Abstract**

In this paper we study the problem of designing an adaptive hash table for redundant data storage in a system of storage devices with arbitrary capacities. Ideally, such a hash table should make sure that (a) a storage device with $x\%$ of the available capacity should get $x\%$ of the data, (b) the copies of each data item are distributed among the storage devices so that no two copies are stored at the same device, and (c) only a near-minimum amount of data replacements is necessary to preserve (a) and (b) under any change in the system. Hash tables satisfying (a) and (c) are already known, and it is not difficult to construct hash tables satisfying (a) and (b). However, no hash table is known so far that can satisfy all three properties as long as this is in principle possible. We present a strategy called SPREAD that solves this problem for the first time. As long as (a) and (b) can in principle be satisfied, SPREAD preserves (a) for every storage device within a $(1 \pm \epsilon)$ factor, with high probability, where $\epsilon > 0$ can be made arbitrarily small, guarantees (b) for every data item, and only needs a constant factor more data replacements than minimum possible in order to preserve (a) and (b).

## 1 Introduction

In this paper, we consider the problem of designing a hash table for a system of heterogeneous storage devices so that the following conditions are met:

1. **Fairness:** Every storage device is assigned a fair share of the total data load, or more precisely, a storage device with $x\%$ of the available capacity should get $x\%$ of the data. Besides the data, the fairness condition should also hold for the read/write requests.

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

3. **Redundancy:** The copies of each data item are distributed among the storage devices so that no two copies are stored on the same device.

4. **Adaptivity:** The system should be able to preserve the conditions above (as long as this is in principle possible) under any change in the system (including changes in the number or size of the storage devices or in the number of data items) with a near-minimum amount of data movements. Adaptivity is measured by applying competitive analysis. That is, for an operation $\omega$ representing any change in the system, we intend to compare the number of (re-)placements of copies performed by the given placement scheme with the number of (re-)placements of copies performed by an optimal strategy that ensures that, after every operation, the fairness and redundancy conditions are satisfied. A storage strategy is called *$c$-adaptive* concerning operation $\omega$ if, on expectation, it requires the (re-)placement of at most $c$ times the number of copies an optimal strategy would need for $\omega$.

Hash tables with these properties have many interesting applications. Consider, for example, a storage system that consists of a large collection of disks. Over the time, old disks have to be replaced by new disks or new disks have to be added in order to provide sufficient capacity to the users. In this case, it would be desirable to just buy any disk available on the market and plug it into the system instead of making sure that it has the same capacity or performance as some other disks in the system (so that standard striping strategies such as those used in the RAID standard can be applied for redundant data storage). In this case, storage strategies are needed that are fair so that capacity or performance differences among the disks can be taken into account. Also redundancy is important for disk systems since disks can fail, and adaptivity ensures that the work for adding or removing disks can be kept small so that the system has a high availability. The adaptivity property

[*] Heinz Nixdorf Institut, University of Paderborn, Germany. Email: vodisek@upb.de

[†] Institut für Informatik, Technische Universität München, Germany. Email: scheideler@in.tum.de.

can also be used to take full control of the overhead of integrating a new disk into the system by controlling the speed at which the disk capacity declared to the system grows from 0 to its actual capacity without ever being in an intermediate or unsafe state. Another nice application would be distributed web caches (like the Akamai system) or peer-to-peer systems. Even though in theory, many designs for peer-to-peer hash tables have focused on distributing the data load evenly among the peers, in practice this will not be the best solution simply because the peers differ in reliability, bandwidth and the storage capacity they can or are willing to offer to the peer-to-peer system.

**1.1 Previous work.** Before we present a hash table that can satisfy all of the conditions above, we present previous work in this area and discuss why it is so difficult to adapt these so that all conditions are met.

**Uniform Capacities.** If all storage devices have the same capacity, it is not difficult to satisfy all conditions above if the only changes allowed in the system are to add or remove storage devices, respectively data items. A prominent strategy here is consistent hashing [8] which (in its basic form) uses two hash functions, $g : U \rightarrow [0, 1)$ and $h : V \rightarrow [0, 1)$, where $U$ is the address space of the data items and $V$ is the address space of the storage devices, or *nodes*. Every data item $d \in U$ is stored at the node $v$ with $h(v)$ being closest from above to $g(d)$ (where $[0, 1)$ is seen as a modulo 1 ring). Other adaptive hash table methods for uniform storage devices have been presented in [3, 15], but like consistent hashing, these are hard to extend to the non-uniform case.

**Non-Uniform Capacities.** The first adaptive hash tables that can handle storage devices of arbitrary capacities were introduced in [4], called SHARE and SIEVE. SHARE, for example, uses two stages to reduce the problem of managing non-uniform nodes to uniform nodes so that consistent hashing can be applied.

Let $n$ be the number of nodes in the system and let $c_1, \ldots, c_n \in [0, 1]$ denote their relative capacities, i.e., $\sum_i c_i = 1$. For a hash table to be adaptive, it has to be able to handle any capacity change from $(c_1, \ldots, c_n)$ to $(c'_1, \ldots, c'_n)$. Certainly, any 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 is true, which has been used to bound the adaptivity of SHARE and SIEVE.

FACT 1.1. *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.*

For any capacity distribution, SHARE and SIEVE are efficient, 2-adaptive and fair within a $(1 \pm \epsilon)$ factor, where the constant $\epsilon > 0$ can be made arbitrarily small [4]. However, none of the two strategies can also be made redundant under all capacity distributions that, in principle, allow a fair distribution of the data. Even the case of 2 copies for each data item is hard. Consider the following simple example.

We have three nodes with capacities $c_1 = 1/2$, $c_2 = 1/4$ and $c_3 = 1/4$, and each data item is supposed to have 2 copies stored at different nodes. For a fair distribution of the copies, we must restrict ourselves to the combinations (node 1, node 2) and (node 1, node 3) for the 2 copies of a data item. If we also allow combinations (node 2, node 3), fairness cannot be achieved any more. Hence, in order to achieve fairness and redundancy at the same time, we have to fight with the problem of forbidding certain combinations (or more generally, to carefully select node combinations and give them appropriate weights). SHARE and SIEVE do not offer an easy way of doing this. In [16], a related strategy, called DHHT, was introduced but this strategy cannot handle redundancy and fairness as well.

**Other Approaches.** In [11], Litwin, *et al.* describe LH* schemes, a class of scalable distributed data structures (SDDS) which are derived from linear hashing [5]. Several LH* variants incorporate fault-tolerance features, such as mirroring, checksum, or Reed-Solomon codes (see e.g. [1, 9, 10, 12]). Depending on the overflow policy, space utilization is more or less fine, but the LH* variants provide no mechanisms to cope with non-uniform disk capacities.

In [6, 7], Honicky and Miller introduce a family of pseudo-random algorithms that provide a decentralized mapping of replicated objects to a collection of disks that can be grouped into clusters of homogeneous disks. These clusters are assumed to be of sufficient size so that the copies of each data item assigned to a cluster can be stored in different disks. In this case, redundancy and fairness can be achieved, but there is no obvious way of extending their schemes to disks of arbitrary capacities so that these properties are still true.

Recently, Brinkmann et al. [2] proposed the first scheme for systems of disks with arbitrary capacities that is fair and redundant, but it 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 $r$.

### 1.2 Our contributions.

In this paper, we introduce a storage strategy named SPREAD with the following performance.

Theorem 1.1. *Consider the problem of storing $r$ copies for each data item in a fair and redundant way in a system of heterogeneous storage devices. SPREAD solves this problem in an efficient way while being $O(1)$-adaptive for any capacity change in the system as long as at any time, $c_i \leq 1/r$ for every node $i$.*

Note that the condition that $c_i \leq 1/r$ for every node $i$ is necessary for being redundant and fair, and therefore SPREAD works in any case in which redundancy and fairness can, in principle, be achieved. The rest of the paper is devoted to the proof of the theorem.

## 2 The SPREAD Strategy

In this section, we describe and analyze the SPREAD strategy. We first describe the basic framework of SPREAD, leaving out various details about how to achieve fairness, redundancy and adaptivity. Afterwards, we bound the time- and space-efficiency of SPREAD (Section 2.2), describe how to achieve fairness and redundancy (Section 2.3), and show how to make SPREAD $O(1)$-adaptive (Sections 2.4 and 2.5). At the end, we discuss how to extend SPREAD to the case that the data items have different levels of redundancy.

### 2.1 The basic scheme.

In the following, $V$ represents the set of all storage system (or node) identifiers and $U$ represents the set of all data identifiers. The parameter $r$ represents the redundancy required for the data items in the system.

For any history of changes in the system up to some given time point, let $n$ be the maximum number of nodes that have been in the system at the same time. When using the strategy that every node newly entering the system obtains the lowest available identifier $\geq 1$, we can make sure that the nodes will be numbered in a range from 1 to $n$. Let $c_1, \ldots, c_n \in [0, 1]$ represent the relative capacities of the nodes at some given time point (i.e., $\sum_i c_i = 1$). (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 needs three (pseudo-)random hash functions: a hash function $h : V \to [0, 1)$ for the nodes and two hash functions $g_1, g_2 : U \to [0, 1)$ for the data items. The interval $[0, 1)$ will be considered as a modulo 1 ring. Like in SHARE [4], SPREAD makes use of a stretch factor $s = \sigma \log N$ where $N = |V|$ (resp. the maximum number of nodes the system can expect) and $\sigma$ is a sufficiently large constant that is chosen so that $s \in \mathbb{N}$. For each node $v$ 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. $h(v)$ is called its *starting point* and $h(v) + s \cdot r \cdot c_i \mod 1$ its *endpoint.* If $s \cdot r \cdot c_i > 1$, we consider $I(v)$ to be wrapped around $[0, 1)$ $\lfloor s \cdot r \cdot c_i \rfloor$ many times.

The SPREAD data structure maintains a partition of the $[0, 1)$ interval into $n$ *frames* $F_1, \ldots, F_n$ where $F_v$ starts at $h(v)$ and ends at the closest successor of $h(v)$ among the points $\{h(1), \ldots, h(n)\} \setminus \{h(v)\}$ (where $[0, 1)$ is treated as a ring). Each frame $F_v$ is further partitioned into *subframes.* We will explain later how these subframes are chosen. For now, we just mention 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, some of these subframes may be partitioned into smaller subframes.

For each subframe $f$, SPREAD aims at maintaining one (and sometimes two, as will be explained later) $(s/\alpha) \times r$-table $T_f$ of $r \cdot s/\alpha$ slots which are organized into $s/\alpha$ *groups* (or columns) of size $r$ each, where $\alpha > 0$ is a small constant that is selected so 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. Then we can use the following strategy in order to ensure redundancy.

Whenever we want to read or overwrite the copies of some data item $d$, we perform the following steps. First, we identify the unique subframe $f$ with $g_1(d) \in f$. Then we pick group $\lceil (s/\alpha) \cdot g_2(d) \rceil$ in $T_f$ and either read or store the copies of $d$ in the $r$ nodes owning the slots in this group.

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.

Lemma 2.1. *The lookup and insert operations of SPREAD can be implemented with runtime $O(r + \log m)$ and space $O(m(r \cdot s/\alpha) \log n)$.*

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.

### 2.2 Subframe management.

Ideally, we would like to have the following subframe decomposition for each frame $F_v$. For the subframes $f_0, f_1, f_2, \ldots$ following $F_v$ in the $[0, 1)$-ring (in ascending direction) it holds that $|f_i| = \epsilon(1 + \epsilon)^i |F_v|$ for some fixed $\epsilon > 0$. If this were true, we could approximate any interval $I(v)$ with $|I(v)| \geq |F_v|$ by an interval $I'(v)$ ending at a starting point of some subframe $f_i$ so that $|I'(v)|$ is within $(1 \pm \epsilon)|I(v)|$. In fact, the following lemma holds.

Lemma 2.2. *Suppose that $k$ is the maximum value so that $|F_v| + \sum_{i=0}^{k-1} |f_i| \leq |I(v)|$. Then $|F_v| + \sum_{i=0}^{k-1} |f_i| \geq (1 - \epsilon)|I(v)|$ and $|F_v| + \sum_{i=0}^{k} |f_i| \leq (1 + \epsilon)|I(v)|$.*

*Proof.* It holds that $|F_v| + \sum_{i=0}^{k-1} |f_i| = (1 + \epsilon \sum_{i=0}^{k-1}(1 + \epsilon)^i)|F_v| = (1 + \epsilon)^k|F_v|$ 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 $|F_v| + \sum_{i=0}^{k-1} |f_i| \geq 1/(1+\epsilon)|I(v)| = (1 - \epsilon/(1+\epsilon))|I(v)| \geq (1 - \epsilon)|I(v)|$ and $|F_v| + \sum_{i=0}^{k} |f_i| \leq (1 + \epsilon)|I(v)|$. □

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 $(1 \pm \epsilon)|I(v)|$. If $|I(v)|$ is at most $|F_v|$, we identify $I'(v)$ with $I(v)$, i.e., we do not round $I(v)$.

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

**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) of the endpoint of $f$ to the starting endpoint of $f'$. For every $w \in \{1, \ldots, n\} \setminus \{v\}$ and every subframe $f$ in $F_v$ we require that $|f| \leq \epsilon(1 + \epsilon)^k|F_w|$ where $k \in \mathbb{N}_0$ is maximum possible so that $\epsilon \sum_{i=1}^{k-1}(1 + \epsilon)^i|F_w| \leq \Delta(F_w, f)$.

If this condition is true, it is easy to see that we can still 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 $(1 \pm \epsilon)|I(v)|$.

In order to satisfy the subframe condition, we use the following decomposition strategy. Given $h(1), \ldots, h(n)$, we start with a 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 $h(1), \ldots, h(n)$) with the property that for every subframe $f$ that is not the first of last subframe in some $F_v$, $|f| = 1/2^k$ for some $k \in \mathbb{N}_0$. Moreover, whenever $n$ is increased, 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 2.3. *The number of frames $F_v$ in the system with $|F_v| \leq \delta/n$ is bounded by $2\delta n + O(\sqrt{n})$, w.h.p., and the smallest size of a frame $F_v$ is at least $1/n^k$ for some constant $k$, w.h.p.*[1]

*Proof.* For every node $v$ let the random variable $X_v$ be equal to $h(v)$. Given a fixed $\delta$, let the function $f(X_1, \ldots, X_n)$ represent the number of frames of size at most $\delta/n$. It certainly holds that $|f(x_1, \ldots, x_n) - f(x'_1, \ldots, x'_n)| \leq 2$ whenever $(x_1, \ldots, x_n)$ and $(x'_1, \ldots, x'_n)$ differ in at most one coordinate. Moreover, it holds for any $v$ and $\delta \leq 1$ that $\Pr[|F_v| \leq \delta/n] = 1-(1-\delta/n)^{n-1} \leq 1-(1-(n-1)\delta/n+\binom{n-1}{2}(\delta/n)^2) = (n-1)\delta/n-\binom{n-1}{2}(\delta/n)^2 \leq \delta$, which implies that $\mathrm{E}[f] \leq \delta n$. Hence, it follows from the method of bounded differences (e.g., [13]) that for any $d \geq 0$, $\Pr[f \geq \delta n + d] \leq e^{-d^2/(2n)}$. Choosing $d = \delta n + \Theta(\sqrt{n})$ results in the first part of the lemma. The second part holds because for any $v$, $\Pr[|F_v| \leq 1/n^k] \leq 1/n^{k-1}$, and therefore the probability that there exists a $v$ with $|F_v| \leq 1/n^k$ is polynomially small in $n$. □

Lemma 2.4. *The number of subframes in the system is bounded by $O(n/\epsilon)$, w.h.p.*

*Proof.* For any $i \in \mathbb{N}_0$, let $S_i$ be the set of all frames with a size in $[2^{-i}/n, 2^{-(i+1)}/n)$. For each frame $F \in S_i$, we need at most $k = \lceil \log_{1+\epsilon}(1/n|F|) \rceil \leq \lceil \log_{1+\epsilon} 2^{i+1} \rceil = O(i/\epsilon)$ subframes $f_0, f_1, \ldots, f_k$ in the ideal setting until $|f_k| = \epsilon(1 + \epsilon)^k|F| \geq \epsilon/n$. Hence, it follows from Lemma 2.3 that the total number of subframes needed is bounded by $\sum_{i=0}^{\log n}(2n/2^i) \cdot O(i/\epsilon) + O(\sqrt{n}) \cdot O(\log n/\epsilon) + O(n/\epsilon) = O(n/\epsilon)$, w.h.p. □

Next, we explain how to manage the tables.

**2.3 Table management.** For each subframe $f$, $C(f)$ represents a multiset of nodes $w$ with $f \cap I(w) \neq \emptyset$. The number of times a node $w$ occurs in $C(f)$ is called its *multiplicity* in $C(f)$ and denoted as $\mu_f(w)$. Initially, $\mu_f(w)$ is set to the number of times $I(w)$ crosses the starting point of $f$, which is in $\{\lfloor r \cdot s \cdot c_i \rfloor, \lceil r \cdot s \cdot c_i \rceil\}$. Hence, given that $c_i \leq 1/r$ for every $i$ (which is necessary to maintain fairness), it holds that $\mu_f(w) \in \{0, \ldots, s\}$. Afterwards, we are using the following rules for every subframe $f \subseteq F_v$ and $w \neq v$:

**Rounding conditions:**

1. Whenever the endpoint of $I(w)$ moves beyond the endpoint of $f$ for the first time after crossing the starting point of $f$, $\mu_f(w)$ is increased by 1.

[1] In the following, "w.h.p." or "with high probability" means a probability of at least $1 - 1/n^c$ for any constant $c > 0$.

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), $\mu_f(w)$ is decreased by 1.

The same rules are also applied 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 $f$. With these rules it holds that whenever the endpoint of $I(w)$ is outside of $f$ then $\mu_f(w)$ is equal to 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, together with the fact that 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 $|C(f)|$. This allows us to prove the following property.

Lemma 2.5. *For every subframe $f$, $|C(f)|$ 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 $\sigma$ in $s$.*

*Proof.* The arguments above imply that, in order to bound $|C(f)|$ for any $f$, it suffices to consider any point $x \in [0, 1)$ representing a starting point or endpoint of some interval $I(v)$. We first consider the starting point of an interval $I(v)$.

Consider some fixed node $v$ with starting point $h(v)$. For any node $w$ in the system (including $v$), let the random variable $X_w$ be defined as $X_w = \lfloor s \cdot r \cdot c_w \rfloor + Y_w$ where the binary random variable $Y_w$ is 1 if and only if $I(w)$ contains $h(v)$ $\lceil s \cdot r \cdot c_w \rceil$ 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 (e.g., [14]) that $\Pr[|Y - \mathrm{E}[Y]| \geq \beta \mathrm{E}[Y]] \leq e^{-\beta^2 \mathrm{E}[Y]/(2(1+\beta/3))}$ 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 $\sigma$ in $s$.

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

For $W = \{1, \ldots, n\}$ let $W_\ell$ be the set of nodes $v \in W$ with $c_v \geq 1/((s-1)r)$ and $W_s = W \setminus W_\ell$. Nodes $v \in W_\ell$ are guaranteed to have $\mu_f(v) \geq 1$ in every subframe $f$. For any subframe $f$, let $C_\ell(f) = C(f)|_{W_\ell}$ (i.e., the multiset containing only those nodes in $C(f)$ that are also in $W_\ell$) and $C_s(f) = C(f)|_{W_s}$. Let $a_\ell = \sum_{v \in W_\ell} c_v$ and $a_s = 1 - a_\ell$. A more refined version of Lemma 2.5 is as follows.

Lemma 2.6. *For any subframe $f$ and any subset $W'_\ell$ of $W_\ell$, $|C'_\ell(f)|$ 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 $\sigma$ 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 $\Pr[|Y - \mathrm{E}[Y]| \geq \beta \mathrm{E}[Y]] \leq e^{-\beta^2 \mathrm{E}[Y]/(2(1+\beta/3))}$ for any $\beta > 0$.

The second bound follows along the same lines as in the proof of Lemma 2.5. □

Furthermore, the following result holds, which is crucial for SPREAD to be redundant.

Lemma 2.7. *For every subframe $f$ there are at least $r$ different nodes in $C(f)$, w.h.p., where the probability depends on the constant $\sigma$ 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 $c_v \leq 1/r$ for all $v$, a relative capacity of at least $1 - q/r$ must be covered by $W_s$. Hence, $|W_s| \geq (1 - q/r)/(1/((s-1)r)) = (s-1)(r-q)$ and $\sum_{v \in W_s} |I(v)| \geq (r \cdot s)(1 - q/r) = s(r - q)$.

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 $X_w$ be 1 if and only if $y \in I'(w)$ and let $X = \sum_w X_w$. Since $\mathrm{E}[X_w] = \Pr[X_w = 1] = \min\{|I'(w)|, 1\}$ for all $w \in W_s \setminus \{v\}$ it holds that

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

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 hat for each subframe $f$, we maintain one (and sometimes two) $(s/\alpha) \times r$-table $T_f$ of $r \cdot s/\alpha$ slots which are organized into $s/\alpha$ *groups* (or columns) of size

$r$ each. We assume that $\alpha > 0$ is a sufficiently small constant with $1/\alpha \in \mathbb{N}$. 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 $s/\alpha$ 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 $\gamma$ that is sufficient to maintain the table conditions is given in the following lemma. In this lemma, $\alpha$ is the parameter used in the table size and $\beta$ is the parameter used in Lemmas 2.5 and 2.6.

LEMMA 2.8. *If the bounds in Lemma 2.6 are true and $\alpha$ and $\beta$ are sufficiently small, then conditions 1 and 2 can be met with any $\gamma \geq \beta/(1-\beta) + \alpha$.*

*Proof.* Recall the bounds in Lemma 2.6 and ignore the $O(\log n)$ terms for a moment. We consider the nodes in $W_\ell$ and $W_s$ separately, starting with $W_\ell$.

Let $W_h \subseteq W_\ell$ be the set of all nodes $v$ with $c_v$ large enough so that it is guaranteed that $\mu_f(v) \geq 1/\gamma$ for $\gamma$ as chosen in the lemma. In this case, every $v \in W_h$ satisfies $(1-\gamma)\mu_f(v)/\alpha \leq (\mu_f(v)-1)/\alpha \leq \lfloor s \cdot r \cdot c_v/\alpha \rfloor$ and $(1+\gamma)\mu_f(v)/\alpha \geq (\mu_f(v)+1)/\alpha \geq \lceil s \cdot r \cdot c_v/\alpha \rceil$. Moreover, $\lceil s \cdot r \cdot c_v/\alpha \rceil \leq s/\alpha$. Hence, each $v \in W_h$ can be given either $\lfloor s \cdot r \cdot c_v/\alpha \rfloor$ or $\lceil s \cdot r \cdot c_v/\alpha \rceil$ many slots without violating condition 1. This implies that the nodes in $W_h$ can be given slots so that the total number of slots used by the nodes in $W_h$ is in $[\lfloor s \cdot r \cdot a_h/\alpha \rfloor, \lceil s \cdot r \cdot a_h/\alpha \rceil]$ where $a_h = \sum_{v \in W_h} c_v$. This is perfect up to an additive 1.

Next, consider the nodes in $W'_\ell = W_\ell \setminus W_h$. Let $C'_\ell(f) = C_\ell(f)|_{W'_\ell}$ be the multiset of nodes $v \in C_\ell(f)$ that are not in $W_h$. In this case, $(1+\gamma)\mu_f(v)/\alpha < s/\alpha$ (given that $s$ is sufficiently large), so we do not have to worry about limiting the number of slots of $v$ by $s/\alpha$. Let $a'_\ell = \sum_{v \in W'_\ell} c_v$. 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 2.6). If $\phi > 0$ is chosen so that $\sum_{v \in W'_\ell}[\mu_f(v)/\alpha + \phi/\alpha] \geq s \cdot r \cdot a'_\ell/\alpha$, then it suffices to assign at most $(\mu_f(v)+\phi)/\alpha + 1$ slots to every node $v \in C'_\ell(f)$ to cover at least an $a'_\ell$-fraction of the slots in $f$. Since $\sum_{v \in W'_\ell} \mu_f(v) = |C'_\ell(f)|$, it holds that $\sum_{v \in W'_\ell}[\mu_f(v)/\alpha + \phi/\alpha] \geq s \cdot r \cdot a'_\ell/\alpha$ 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 $\phi > 0$ is chosen so that $\sum_{v \in W'_\ell}[\mu_f(v)/\alpha - \phi/\alpha] \leq s \cdot r \cdot a'_\ell/\alpha$, then at least $(\mu_f(v)-\phi)/\alpha - 1$ slots can be assigned to every node $v \in C'_\ell(f)$ to cover at most an $a'_\ell$-fraction of the slots in $f$. For this we have to choose $\phi$ so that $\phi \geq \beta$. 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 $(\mu_f(v)+\beta)/\alpha + 1 \leq (1+\gamma)\mu_f(v)/\alpha$ and $(\mu_f(v)-\beta)/\alpha - 1 \geq (1-\gamma)\mu_f(v)/\alpha$.

Finally, consider the nodes in $W_s$. 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 2.6). If $\phi > 0$ is chosen so that $\sum_{v \in W_s}(1+\phi)\mu_f(v)/\alpha \geq s \cdot r \cdot a_s/\alpha$, then it suffices to assign at most $(1+\phi)\mu_f(v)/\alpha + 1$ slots to every node $v \in C_f$ to cover an $a_s$-fraction of the slots in $f$. Since $\sum_{v \in W_s} \mu_f(v) = |C_s(f)|$, we have to choose $\phi$ so that $(1+\phi)(1-\beta)s \cdot r \cdot a_s/\alpha \geq s \cdot r \cdot a_s/\alpha$, which works for $\phi \geq 1/(1-\beta) - 1 = \beta/(1-\beta)$. For a lower bound, suppose that $|C_s(f)| = (1+\beta)s \cdot r \cdot a_s$. If $\phi > 0$ is chosen so that $\sum_{v \in W_s}(1-\phi)\mu_f(v)/\alpha \leq s \cdot r \cdot a_s/\alpha$, then at least $(1-\phi)\mu_f(v)/\alpha - 1$ 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 $\phi$ so that $(1-\phi)(1+\beta)s \cdot r \cdot a_s/\alpha \leq s \cdot r \cdot a_s/\alpha$ which works for $\phi \geq 1 - 1/(1+\beta) = \beta/(1+\beta)$. Hence, $\gamma \geq \beta/(1-\beta) + \alpha$ is sufficient for both cases.

Finally, notice that there is this $O(\log n)$ term in the bounds in Lemma 2.6 that we ignored so far. However, whenever this term is dominant for a set of nodes in some subframe, the sum of their capacities is at most $\delta$ for some constant $\delta > 0$ that can be made arbitrarily small depending on $\sigma$. Hence, since we are considering only three sets of nodes, those sets of nodes where the $O(\log n)$ term is negligible (and can therefore be covered by $\beta$) 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 $\delta$ is sufficiently small compared to $\alpha$ and $\beta$. □

If conditions 1 and 2 are true, also condition 3 can be met. To show this, consider the slots to be numbered row-wise from 1 to $r \cdot s/\alpha$ by giving slot $(i,j)$ in group $i$ the number $(j-1) \cdot s/\alpha + i$. Assign the slots to the nodes so that each node $v \in C(f)$ owns a consecutive sequence of slots. Since every node owns at most $s/\alpha$ slots, no group can have two slots owned by the same node, which proves our claim. The challenge, of course, will be to maintain these conditions as the system changes without rearranging too many slot assignments. Before we show how to do this, we prove that the table conditions allow us to maintain fairness.

LEMMA 2.9. *If the table conditions are met, then for* *every node $v$ in the system and every data item $d$,* $\Pr[v$ *stores a copy of* $d]$ *is in* $[(1-\gamma)(1-\epsilon)rc_v, (1+\gamma)(1+\epsilon)rc_v]$ *for the $\gamma$ in condition 2 and $\epsilon$ as chosen for the interval rounding.*

*Proof.* Consider any node $v$ and data item $d$. Let $p = |I'(v)| \mod 1$, where $I'(v)$ is the rounded form of $I(v)$. For an area of size $p$ in $[0,1)$, $v$ has a multiplicity of $\lceil |I'(v)| \rceil$, and for the remaining area of size $(1-p)$ in $[0,1)$, $v$ has a multiplicity of $\lfloor |I'(v)| \rfloor$. Moreover, it follows from the table conditions that for any node $v$ in some subframe $f$, $\Pr[v$ selected by a data item$] \in [(1-\gamma)\mu_f(v)/s, \min\{(1+\gamma)\mu_f(v), s\}/s]$. Hence, it holds for data item $d$ that

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

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

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

**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.

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 2.9 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.

Suppose that the capacities of the nodes change from $c_1, \ldots, c_n$ to $c'_1, \ldots, c'_n$. For each node identifier $v$ that has not been used before (i.e., $n$ increases), some subframes may have to be cut into smaller subframes that take over the tables of the previous subframes. Hence, if the previous subframes satisfied the table conditions, the new ones will also do so. Our rounding conditions may then require updates to these tables, which can be charged to past capacity changes as for the other cases below. Afterwards, we start with adapting the intervals $I(v)$ to $c'_1, \ldots, c'_n$. This may cause changes in $C(f)$ of a subframe $f$, which may require updates in its table (or tables). Let us call a subframe $f$ *dirty* if it is in $F_v$, contains the endpoint of $I(v)$ and $|I(v)| \leq |F_v|$. Otherwise, it is called *clean.* First, we 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. If $C'(f) \neq C(f)$, we go through the following stages.

1. **Pairing stage:** Suppose that the total decrease in the multiplicities of nodes in $C(f)$ is $\delta_d$ and the total increase in the multiplicities of nodes in $C(f)$ is $\delta_i$. 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 $\mu_f(v) := \mu_f(v) - 1$ and $\mu_f(w) := \mu_f(w) + 1$ and then change slot assignments until table condition 2 is satisfied for $v$ and $w$. For each such slot reassignment, we distinguish between three cases. If both $v$ and $w$ violate condition 2, then a slot of $v$ is given to $w$. 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.). 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 $x$ 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 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 $s/\alpha$ 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 a node $u''$ that has 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 update $v$'s multiplicity 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 current multiset during the process of moving from $C(f)$ to $C'(f)$.

Lemma 2.10. *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, 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, $s_v < (1-\gamma)\mu_f(v)/\alpha$. As long as there is a node $w$ with $s_w - 1 \geq (1-\gamma)\mu_f(w)/\alpha$, we can move a slot from $w$ to $v$ until $s_v \geq (1-\gamma)\mu_f(v)/\alpha$. 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 $s_v < (1-\gamma)\mu_f(v)/\alpha$ but there is no node $w$ any more with $s_w - 1 \geq (1-\gamma)\mu_f(w)/\alpha$. In this case, the total number of slots occupied by the nodes in $C''(f)$ is less than

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

If $\gamma \geq \beta/(1-\beta) + \alpha$, it follows from Lemma 2.5 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. It holds that $(1-2\beta)(1+\beta) = 1-\beta-2\beta^2 < 1-\beta$, so $\sum_w s_w < s \cdot r/\alpha$, 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 $v$ in this case.

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

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

w.h.p. This, however, is a contradiction. Hence, slots from $v$ 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 2.11. *In each step of the pairing or movement stage, at most $2(1+\gamma)/\alpha$ 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. We know that for the old multiplicity $\mu_f(v)$ of $v$ it holds that $s_v \geq (1-\gamma)\mu_f(v)/\alpha$. Hence, at most $1/\alpha$ slots have to be moved to $v$ to satisfy $s_v \geq (1-\gamma)(\mu_f(v)+1)/\alpha$. Since each slot movement may require a flip with another slot to repair condition 3, the total number of slot reassignments is at most $2/\alpha$ in this case.

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

Next, consider a step of the pairing stage. Suppose that the multiplicity of node $v$ decreases by 1 while the multiplicity of $w$ increases by 1. Then $v$ has to give up at most $(1+\gamma)/\alpha$ slots while $w$ has to get at most $1/\alpha$ 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 $2(1+\gamma)/\alpha$ slot reassignments are needed to repair condition 2 for $v$ and $w$. □

Hence, given a total change in the multiplicities of the nodes by $\mu$, at most $2\mu(1+\gamma)/\alpha$ 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 $d$ with $g_1(d) \in f$ needs to be replaced is equal to $k/(r \cdot s/\alpha)$. Thus, the expected number of copy movements is at most

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

where $D \subseteq U$ is the set of data items that are stored in the system. A change in multiplicities by $\mu$ can be charged to a capacity change of $c(f) \geq |f|\mu/(s \cdot r)$ 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 1.1, a capacity change of $c(f)$ requires the replacement of at least $c(f) \cdot r|D|/2$ 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 only difference that $C(f_1)$ contains a copy of node $v$ while $C(f_2)$ does not contain $v$. Our goal is to make sure that $T_1$ and $T_2$ differ in at most $2(1+\gamma)/\alpha$ 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 the table for $f_1$ is obtained by applying slot reassignments for $v$ to $T_2$ until the table conditions are met for $f_1$. According to Lemma 2.11, this requires the reassignment of at most $2(1+\gamma)/\alpha$ 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 $2(1+\gamma)/\alpha$ further slot reassignments in order to remove or add slots for $v$. 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 $\ell$ in $f$. Then this can be charged to a change in capacity of $v$ of $c = \ell/(s \cdot r)$. Hence, the proximity condition ensures that the expected number of copy movements is at most

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

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 $\ell$ 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 $|f_1|$ is larger than $|f_2|$. Then at most $2\mu(1+\gamma)/\alpha$ slots are reassigned in $T_1$ and at most $(\mu+2)2(1+\gamma)/\alpha$ slots are reassigned in $T_2$. The latter bound holds because $T_2$ differs in at most $2(1+\gamma)/\alpha$ slots from $T_1$ before and after the reassignments. Since a total change of $\mu$ can be charged to a capacity change of $c(f) \geq |f|\mu/(s \cdot r)$ 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

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

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 $8(1+\gamma)$.

**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 $h' : V \rightarrow [0,1)$, 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 $x + h'(v)/|f|$ mod 1 from below, $\mu_f(w)$ is increased by 1.

2. Whenever the endpoint of $I(w)$ crosses $x + h'(v)/|f|$ mod 1 from above, $\mu_f(w)$ is decreased by 1.

With this rule we obtain the following result:

Lemma 2.12. *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 $\mu_f(v)$ increases or decreases by 1 is equal to $|\delta I(v)|/|f|$. 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 $(4(1+\gamma)|f|/s) \cdot |D|$ copies in the system. Since $c_v$ changed by $\delta c_v = |\delta I(v)|/(r \cdot s)$, this means that the expected number of copies replaced due to $v$ is at most $(|\delta I(v)|/|f|) \cdot (4(1+\gamma)|f|)/s \cdot |D| = 4(1+\gamma)\delta c_v \cdot r|D|$. For $\delta I(v) \not\subseteq f$, similar arguments also yield that the expected number of copies replaced due to $v$ is at most $4(1+\gamma)\delta c_v \cdot r|D|$.

If $n$ 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 $x + h'(v)/|f|$ or $y$, or if $I'(v) \subset I(v)$ and the endpoint of $I(v)$ passes $x$ or $x + h'(v)/|f|$, which preserves our adaptivity bound. □

Combining all of the results in this section yields Theorem 1.1. 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 $\varphi$ 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.

**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 $r$.

## References

[1] LH*$_{\mathrm{RS}}$: A high-availability scalable distributed data structure using reed solomon codes. In *SIGMOD Conference*, pages 237–248, 2000.

[2] A. Brinkmann, S. Effert, F. M. auf der Heide, and C. Scheideler. Dynamic and redundant data placement. In *Proc. of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2007.

[3] A. Brinkmann, K. 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.

[4] A. Brinkmann, K. 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.

[5] R. J. Enbody and H. C. Du. Dynamic hashing schemes. *ACM Comput. Surv.*, 20(2):850–113, 1988.

[6] R. Honicky and E. Miller. A fast algorithm for online placement and reorganization of replicated data. 2003.

[7] R. J. Honicky and E. L. Miller. Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution. In *Proceedings of the 18th IPDPS Conference*, 2004.

[8] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. of the 29th ACM Symposium on Theory of Computing*, pages 654–663, 1997.

[9] W. Litwin, J. Menon, and T. Risch. LH* schemes with scalable availability. Technical Report RJ 10121 (91937), IBM Research, Almaden Center, May 1998.

[10] W. Litwin and M.-A. Neimat. High-availability LH* schemes with mirroring. In *Conference on Cooperative Information Systems*, pages 196–205, 1996.

[11] W. Litwin, M.-A. Neimat, and D. A. Schneider. LH* - a scalable, distributed data structure. *ACM Trans. Database Syst.*, 21(4):480–525, 1996.

[12] W. Litwin and T. Risch. LH*$_{\mathrm{G}}$: A high-availability scalable distributed data structure by record grouping. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):923–927, 2002.

[13] 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.

[14] C. McDiarmid. Concentration. In M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, pages 195–247. Springer Verlag, Berlin, 1998.

[15] 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.

[16] C. Schindelhauer and G. 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.