Alcom-FT Technical Report Series
ALCOMFT-TR-02-124

Compact, Adaptive Placement Schemes for Non-Uniform Capacities

André Brinkmann*
Heinz Nixdorf Institute and
Dept. of Electrical Engineering
University of Paderborn
33102 Paderborn, Germany
brinkman@hni.upb.de

Kay Salzwedel †
Heinz Nixdorf Institute and
Dept. of Mathematics and
Computer Science
University of Paderborn
33102 Paderborn, Germany
kay@hni.upb.de

Christian Scheideler‡
Dept. of Computer Science
Johns Hopkins University
3400 N. Charles Street
Baltimore, MD 21218, USA
scheideler@cs.jhu.edu

ABSTRACT

In this paper we study the problem of designing compact, adaptive strategies for the distribution of objects among a heterogeneous set of servers. Ideally, such a strategy should allow the computation of the position of an object with a low time and space complexity, and it should be able to adapt with a near-minimum amount of replacements of objects to changes in the capabilities of the servers so that objects are always distributed among the servers according to their capabilities. Previous techniques are able to handle these requirements only in part. For example, standard hashing techniques can be used to achieve a non-uniform distribution of objects among a set of servers and the time and space efficient computation of the position of the objects, but they usually do not adapt well to a change in the capabilities. We present two strategies based on hashing that achieve all of the three goals. Furthermore, we give a list of applications for these strategies demonstrating that they can be used efficiently for distributed data management, web caches, and peer-to-peer networks.

1. INTRODUCTION

In this paper we study the problem of designing compact, adaptive placement schemes for servers with non-uniform capacities. These capacities are not only bound to space requirements but can represent any restricting property of the servers. In particular, we are interested in schemes that allow to achieve the following goals:

1. **Faithful distribution**, i.e. distributing a set of objects among a set of servers in such a way that the fraction of objects stored at a server is equal (or at least close) to its share of the total capacity of the system.
2. **Efficient localization**, i.e. computing the position of an object with a low time and space complexity.
3. **Fast adaptation**, i.e. adapting to changing capacities with a near-minimal movement of objects.

Especially the space complexity is an important aspect, since if enough space were available to store a lookup table for all objects, the tasks above could be solved in a trivial way. A placement scheme with a low space complexity is called *compact*.

A standard approach for achieving the first two items has been to use random hash functions. However, the problem with using hash functions is that they are usually not adaptive. Consider, for example, the hash function $h(x) = ((a \cdot x + b) \bmod p) \bmod n$ that can be used to distribute a set of objects among n servers. If a new server is added, we are left with two choices: either replacing n by $n+1$ (which would require virtually all objects to be replaced) or adding additional rules to $h(x)$ to force a certain set of objects to be replaced to the new server (which, in the long run, would destroy the compactness of the hashing scheme).

Compact, adaptive placement schemes have many applications. Consider, for example, a storage system that consists of a large collection of disks. Over the time, new disks may be added and old disks may be taken out or fail. To ensure that a storage system can be used at maximum performance, it would be desirable to distribute the data among the available disks according to their capabilities. As the disks in the system or its configuration change, one has to redistribute data in an efficient way. Techniques currently used in practice such as the various RAID levels cannot solve this problem efficiently. For example, virtually all data has to be replaced to fully integrate a new disk into an existing RAID array. The RAID levels also do not allow to support non-uniform disks. Even though there are some efforts under way to expand the RAID levels to handle non-uniform disks efficiently [8, 3, 2], a general solution has yet to be

*Supported in part by the DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen".

†Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

‡Part of the work was done being a member of the Heinz Nixdorf Institute at the Paderborn University, supported by the DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen"

SPAA'02, August 10-13, 2002, Winnipeg, Manitoba, Canada.

found. In addition to (re-)distributing data among disks according to their capabilities, it is also important to be able to determine the disk storing a particular data item in a fast and compact way to ensure that a high load of data requests can be handled with a reasonable amount of resources.

The problem of distributing blocks among a set of disks is closely related to the application of web caching, i.e. evenly distributing a number of copies of an original object among a set of web caches (see e.g. [7]). Web caching introduces a new level of complexity for the placement algorithm, because in the distributed internet environment the client may not know all participating web caches. One additional task of the algorithm in web caching is to minimize the number of copies of an object that are required to guarantee w.h.p. that if the client has a sufficiently consistent view of the web caches, at least one copy of an object can be found by the client.

Another important application of adaptive placement schemes is the problem of distributing tasks among processors of a distributed system. Adaptivity is important here, since it may not be predictable how long a task has to be executed and how long and to which extent a processor may be available. Hence, in order to exploit the full strength of a distributed system, it may be necessary to redistribute tasks among the processors. Since tasks may need to exchange information, it is also important that the current position of a task can be computed in a fast and compact way.

A fourth important area in which adaptive placement schemes may be a valuable tool are peer-to-peer networks. Different peers may have different computational power or different connections to the Internet (dial-up modems or DSL). Hence, in order to fully exploit the capabilities of a peer-to-peer network, it is desirable to be able to assign different node degrees to different peers. Thus, peers with a fast connection to the Internet, for example, would get more connections to other peers than peers with a slow connection, and therefore also a higher load of requests. This would not only allow to shift more work to more capable peers, but it would also allow to speed up operations used on top of the peer-to-peer networks such as searching.

We will give more applications in Section 4, but first we specify our model and give an overview of previous results and our new results.

1.1 The Model

We adopt and extend the standard balls into bins model in which each server is represented by a bin and each object (representing, e.g., a data block or task) is represented by a ball. Let $\{1,\dots,N\}$ be the set of all possible bins and $\{1,\dots,M\}$ be the set of all possible balls that can be in the system at any time. Suppose that the current number of balls in the system is $m \leq M$ and that the current number of bins in the system is $n \leq N$. We will often assume for simplicity that the balls and bins are numbered in a consecutive way starting with 1 (but any numbering that gives unique numbers to each ball and bin would work for our strategies). Let the current *capacity of bin i* be given by a parameter d_i and the current *capacity distribution of the system* be defined as $(d_1,\dots,d_n)$. The capacity d_i of a bin i can be arbitrarily defined, e.g. as its storage capacity, its bandwidth, its computational power, or a mixture of theses parameters. In the remainder of the paper we will use relative values for the d_i's, i.e. $d_i \in [0,1]$ for each i and $\sum_i d_i = 1$. This can be achieved by simply dividing the original capacity of a bin by the capacity of the system. Our goal is to achieve that every bin i with capacity d_i obtains $d_i \cdot m$ of the balls.

The system may now change in a way that the number of available balls, the number of available bins, or the capacities of the bins change. In this case a placement scheme is needed that fulfills several criteria:

- **Faithfulness:** A scheme is called *faithful* if the expected number of balls it places at bin i is within $(1 \pm \epsilon)d_i \cdot m$ for all i, where $\epsilon > 0$ can be made arbitrarily small.
- **Time Efficiency:** A scheme is called *time-efficient* if it allows a fast computation of the position of a ball.
- **Compactness:** We call a scheme *compact* if the amount of information the scheme requires to compute the positions of the balls is small (in particular, it should only depend on N and m in a logarithmic way).
- **Adaptivity:** We call a faithful scheme *adaptive* if in the case that there is any change in the number of balls, bins, or the capacities of the system, it allows to redistribute balls to get back to a faithful distribution. To measure the adaptivity of a placement scheme, we use competitive analysis. For any sequence of operations σ that represent changes in the system, we intend to compare the number of (re-)placements of balls performed by the given scheme with the number of (re-)placements of balls performed by an optimal strategy that ensures that, after every operation, the distribution of balls among the bins is perfectly faithful. A placement strategy will be called *c-competitive* if for any sequence of changes σ, it requires the (re-)placement of (an expected number of) at most c times the number of balls an optimal adaptive and perfectly faithful strategy would need.

To clarify the last definition, notice that when the capacity distribution in the system changes from $(d_1,...,d_n)$ to $(d'_1,...,d'_n)$, an optimal, perfectly faithful strategy would need only

$$\sum_{i:d_i>d'_i} (d_i - d'_i) \cdot m$$

replacements of objects. Thus, if for example the capacity distribution changes from $(1/2, 1/2, 0)$ to $(0, 1/2, 1/2)$ (bin 1 leaves and bin 3 enters the system), ideally only a fraction of $1/2$ of the objects would have to be moved. We will see in the following sections that the constant ϵ that describes the faithfulness of our algorithms influences the time efficiency and the compactness of our algorithms, but has only a very minor influence on the number of replacements of balls if it is small. Therefore, we will compare the number of replacements of our algorithms with the ideal bound on the replacements of an optimal algorithm above.

1.2 Previous results

Compact, adaptive placement strategies are relatively new. So far, only good strategies are known for uniform capacities, that is, all available bins have the same capacity. In this case, it only remains to cope with situations in which new bins enter or old bins leave the system. Karger et al. [4] present an adaptive hashing strategy that is faithful and 2-competitive (resp. 1-competitive if new bins can be renamed to take over the role of departing bins, but this would destroy several properties shown for it in [4], since it would not be oblivious any more). In addition, the computation of the position of a ball takes only an expected number of $O(1)$ steps. However, their data structures need at least $n \log^2 n$ bits to ensure that with high probability the distribution of the balls does not deviate by more than a constant factor from the desired distribution. Furthermore, extending this strategy in a straight-forward way to achieve a faithful distribution also for the heterogeneous case may require a tremendous increase in the space complexity (see Section 2.1). Brinkmann et al. [1] presented an alternative, 2-competitive placement strategy for uniform demands. Their scheme requires $O(n \log n)$ bits and $O(\log n)$ steps to evaluate the position of a ball. Furthermore, it keeps the deviation from the desired number of balls in a bin extremely small with high probability: if the number of balls fulfills $m \geq n \ln n$, then the maximum number of balls per bin is bounded by $m/n + O(\sqrt{m \ln n/n})$, w.h.p. (The scheme in [4] only achieves $O(m/n)$ with high probability if $O(n \log^2 n)$ bits are used, even if $m \gg n$.) Also the scheme by Brinkmann et al. does not seem to be extendable in a straight-forward way to non-uniform capacities (see [1] for some attempts). Another adaptive placement strategy was proposed by Sanders [6]. He considers the case that bins fail and suggests to use a set of forwarding hash functions $h_1, h_2, \ldots$, where at the time h_i is set up, only bins that are intact at that time are included in its range. From his description it seems that this strategy can cope reasonably well with failed disks, but it runs into problems when the number of disks grows.

1.3 New results

To the best of our knowledge, all three strategies above do not seem to be extendable to non-uniform demands without either a significant increase in memory, losing the faithfulness condition, and/or a bad adaptivity (for further explanations, see Section 2.1). Instead, we found two new strategies, called SHARE and SIEVE, that are compact, faithful for arbitrary non-uniform capacity distributions, and $(2+\epsilon)$-competitive for arbitrary changes from one capacity distribution to another, where $\epsilon > 0$ can be made arbitrarily small. We also demonstrate that these strategies have many interesting applications in distributed data management, web caching, and peer-to-peer networks by proving additional properties of SHARE and SIEVE important for these areas.

In the next two sections we will present and analyze our new strategies SHARE and SIEVE. Afterwards, we give some applications of these strategies, followed by a conclusion.

2. THE SHARE STRATEGY

The SHARE strategy works in two phases. The first phase handles the heterogeneous capacities of the bins by using a data structure (containing intervals of appropriate length for each bin) that allows to reduce the non-uniform placement problem to the uniform placement problem. Given a ball b, the result of this phase is a set S of bins that are equally likely to contain b. In the second phase we can then use a strategy for uniform capacities to determine the bin in S that stores b. Hence, the SHARE strategy requires as a subroutine an adaptive hashing strategy for uniform capacities. We say that a system has *uniform capacities* if for all available bins i, $d_i = 1/n$. In the uniform case, capacities can only change when new bins enter the system or old bins leave the system. Several strategies as noted in the previous results section have already been presented that are competitive for the uniform case. We will give as an example a strategy that was presented by Karger et al. [4], since it will be the most useful for our goals.

2.1 The NEAREST NEIGHBOR strategy

We start with a strategy, called here NEAREST NEIGHBOR, that solves the problem of redistributing balls under uniform capacities. It works as follows:

Suppose that we have a random function f_B and a set of independent, random functions $g_1, \ldots, g_k$, where k may depend on n. The function $f_B : \{1, \ldots, M\} \to [0, 1)$ maps the balls uniformly at random to real numbers in the interval $[0, 1)$ and each function $g_i : \{1, \ldots, N\} \to [0, 1)$ maps the bins uniformly at random to real numbers in the interval $[0, 1)$. Ball i is assigned to the bin closest to it with regard to this mapping when viewing $[0, 1)$ as a ring, i.e. it is assigned to the bin b that minimizes $\min_j \min[|f_B(i) - g_j(b)|, 1 - |f_B(i) - g_j(b)|]$.

From the proofs in [4] it follows that this strategy is faithful and 2-competitive (in the expected sense, if renaming of bins is not allowed) and that it can be implemented in a way that the location of a ball can be determined in expected constant time. It requires k to be $\Omega(\log N)$ to ensure that for any $n \leq N$ it is very unlikely to have a more than constant factor deviation from the expected number of balls in a bin. This results in a space consumption of around $O(n \log N)$ words (assuming that one word can hold $\log n$ bits and not including the hash functions).

One might think that this strategy can be easily extended to cover the heterogeneous case by allowing disks with more capacity to have more random points in $[0, 1)$. However, this would require $\Omega(c_{\max}/c_{\min})$ points to be faithful, where $c_{\max}$ is the maximum capacity and $c_{\min}$ is the minimum capacity of a bin. Thus, in the worst case the number of points could be as much as $\Theta(m)$, violating severely our conditions on the space complexity.

2.2 The SHARE strategy

Now we are ready to describe the SHARE strategy. As mentioned above, SHARE needs an adaptive strategy for uniform capacities. For convenience, we will use the NEAREST NEIGHBOR strategy, which will be called by the command NEAREST(b, S). In this call, b is the number of a ball and S represents the set of bins to which NEAREST NEIGHBOR is applied. The return value of the function is the number of the bin storing b. SHARE is based on two hash functions (in addition to the hash functions that are used by NEAREST NEIGHBOR): a hash function $h : \{1, \ldots, M\} \to [0, 1)$ that maps the balls uniformly at random to real numbers in the interval $[0, 1)$, and a hash function $g : \{1, \ldots, N\} \to [0, 1)$ that maps starting points of intervals for the bins uniformly at random to real numbers in $[0, 1]$. In addition, two fixed parameters $s \geq 1$ and $1/N \leq \delta < 1$ are used. s is the stretch factor needed to ensure high probability and δ is used to simplify the analysis. We will specify their values later. SHARE works in the following way:

Suppose that the capacities for the n given bins are represented by $(d_1, \ldots, d_n) \in [0,1)^n$. To make sure that bins will not have a too high capacity (which would complicate some of our proofs), we use the following strategy: For every bin i with $d_i \geq \delta$, we introduce $\lfloor d_i/\delta \rfloor$ virtual bins i' with $d_{i'} = \delta$ and, if necessary, one additional bin taking the rest so that their total capacity is equal to d_i. Every other bin is left as it is and regarded as a single virtual bin. It is easy to see that this transformation creates $n' \leq n + 1/\delta$ virtual bins with a capacity of at most δ each. Now, every virtual bin i is given an interval I_i of length $s \cdot d_i$, for some fixed *stretch factor* s, that reaches from $g(i)$ to $(g(i)+s\cdot d_i) \bmod 1$, where $[0,1)$ is viewed as a ring. We will assume that $\delta \leq 1/s$ to prevent an interval being wrapped around $[0,1)$ for several times. This keeps the description and analysis of SHARE simple.

For every $x \in [0,1)$ let $C_x = \{i : x \in I_i\}$ and $c_x = |C_x|$, which is called the *contention* at point x. Since the total number of endpoints of all intervals I_i is at most $2n' \leq 2(n+1/\delta)$, $[0,1)$ has to be cut into at most $2(n+1/\delta)$ *frames* $F_j \subseteq [0,1)$ so that for each frame F_j, C_x is the same for each $x \in F_j$. This is important to ensure that the data structures for SHARE have a low space complexity. The computation of the position of a ball b is now simply done by calling UNIFORM$(b, C_{h(b)})$.

Algorithm SHARE(b)**:**
Input: number b of a ball and a data structure containing all intervals I_i
Output: bin number $\in \{1,\ldots,n\}$ that stores b
Phase 1: query data structure for point $h(b)$ to derive the interval set $C_{h(b)}$
Phase 2: bin_b = NEAREST$(b, C_{h(b)})$
return bin_b

Figure 1: The SHARE algorithm.

For this strategy to work correctly, we require that every point $x \in [0,1)$ is covered by at least one interval I_i w.h.p. The next lemma clarifies for which s this is the case.

LEMMA 2.1. *A stretch factor $s = l \cdot \ln N$ with $l \geq 3$ is sufficient to ensure for every $n \leq N$ that w.h.p. $c_x > 0$ for every $x \in [0,1)$ and therefore every ball can be placed.*

PROOF. For every s and i and every point $x \in [0,1)$ it holds that $\Pr[x \in I_i] = s \cdot d_i$. Hence, $\mathrm{E}[c_x] = \sum_i s \cdot d_i = s$. Since the probabilities for the I_i are independent, we can use the Chernoff bounds with $s = l \cdot \ln N$ to show that the probability of a point x having a contention of $c_x = 0$ is

$$\Pr[c_x = 0] = Pr[x = (1-1)\cdot \mathrm{E}[c_x]] \leq \mathrm{e}^{-s/2} = \frac{1}{N^{l/2}}.$$

Having at most $4N$ frames (recall that we require $\delta \geq 1/N$) and therefore at most $4N$ points to consider, the probability that there is at least one frame with a contention of 0 is at most $\frac{4}{N^{l/2-1}}$. □

Next we state a lemma about the time and space complexity of SHARE. We assume that a word can hold $\log(\max[N,M])$ bits. We exclude considering the time and space complexity of the hash functions. Here, simply any efficient hash function out of the vast pool of known hash functions may be chosen.

THEOREM 2.2. *Suppose that the number of hash functions used in NEAREST NEIGHBOR is k. Then SHARE can be implemented so that the position of a ball can be determined in expected time $O(1)$ using a space of $O(s\cdot k\cdot(n+1/\delta))$ words (without considering the hash functions).*

PROOF. The proof for the time complexity uses a trick given in [4]. The idea is to divide $[0,1)$ into *segments* of size $\min[1/n, \delta]$ and to keep a separate search structure for each segment. In this case, the time to locate a ball is equal to the time to compute its segment (which is $O(1)$) plus the time for finding its right frame F (which would give us $C_{h(b)}$) within the segment and the time for executing UNIFORM$(b, C_{h(b)})$. Since the total number of frames is at most $n + 1/\delta$, the expected number of frames overlapping with a segment is constant. Since the expected time for NEAREST$(b, C_{h(b)})$ is also a constant, the total time to locate a ball is a constant.

Concerning the space requirement: As mentioned in the proof of the previous lemma, $\mathrm{E}[c_x] = s$ for every $x \in [0,1)$. Hence, for every beginning and endpoint x of an interval I_i, the expected number of other intervals crossing x is at most s. Considering the fact that no interval starts or leaves within a frame but only at its borders, this implies that the expected number of intervals in a frame F is at most $s+1$. Since there are at most $2(n+1/\delta)$ different frames, the expected amount of words necessary for storing the set of intervals belonging to the frames is $O(s(n+1/\delta))$. Furthermore, $O(n+1/\delta)$ words are necessary to store a data structure for the $\max[n, 1/\delta]$ segments. Finally, NEAREST NEIGHBOR needs for each frame F_j with ℓ_j intervals $O(\ell_j \cdot k)$ words. Since $\mathrm{E}[\ell_j] \leq s+1$, the total amount of space needed by SHARE is $O(s \cdot k \cdot (n+1/\delta))$ words. □

Next we show that SHARE is faithful. For simplicity, we will treat the virtual bins as the real bins, i.e. $n' = n$. Let the *share* S_i of bin i be defined as

$$S_i = \int_{x=g(i)}^{g(i)+s\cdot d_i} \frac{1}{c_x}\, dx \;.$$

S_i has the following property.

LEMMA 2.3. *For any $0 < \epsilon < 1$ it holds: If $s \geq (6 \ln N)/\sigma^2$ with $\sigma = \epsilon/(1+\epsilon)$, then $S_i \in [(1-\epsilon)d_i, (1+\epsilon)d_i]$ for all i w.h.p.*

PROOF. We know from Lemma 2.1 that $\mathrm{E}[c_x] = s$ for all x. Furthermore, the Chernoff bounds imply that for any $0 < \epsilon < 1$,

$$\Pr[c_x \leq (1-\epsilon)\cdot s] \leq \mathrm{e}^{-\epsilon^2 \cdot s/2}$$

and

$$\Pr[c_x \geq (1+\epsilon)\cdot s] \leq \mathrm{e}^{-\epsilon^2 \cdot s/3} \;.$$

Having at most $4N$ intervals and setting $s = (6 \ln N)/\sigma^2$ with $\sigma = \epsilon/(1+\epsilon)$, it follows that with probability at least

$1 - 4N \cdot 2\mathrm{e}^{-\sigma^2 \cdot s/3} \geq 1 - 8/N$,

$$\begin{aligned} S_i &= \int_{x=g(i)}^{g(i)+s\cdot d_i} \frac{1}{c_x}\, dx \leq \int_{x=0}^{s\cdot d_i} \frac{1}{(1-\sigma)\cdot s}\ dx \\ &= \frac{d_i}{(1-\sigma)} \leq (1+\epsilon)d_i \end{aligned}$$

and similarly $S_i \geq (1-\epsilon)d_i$ for all i. □

This allows us to prove the following theorem.

THEOREM 2.4. *If $s = \Omega(\ln N)$, then SHARE is faithful.*

PROOF. Recall that NEAREST NEIGHBOR is faithful if each bin has $k = \Omega(\ln N)$ points, i.e. given n' bins and m' balls, the expected number of balls in bin i is within $(1 \pm \epsilon')d_i \cdot m'$, where ϵ' can be brought arbitrarily close to 0. Now, let the random variable B_x denote the number of balls b with $h(b) = x$ (or more precisely, with $h(b) \in [x, x+dx]$ for $dx \to 0$). For any bin i and any $x \in [g(i), g(i) + s \cdot d_i)$ let $S_x = 1/c_x$ (which is well-defined since $c_x \geq 1$) and B_x^i denote the number of balls in B_x that are assigned to bin i. Furthermore, let the random variable L_i denote the load, i.e. the total number of balls, placed in i. It holds that $L_i = \int_{x=0}^{s\cdot d_i} B_{g(i)+x}^i$. Since UNIFORM is faithful, it holds for all $x \in I_i$ that

$$\begin{aligned} \mathrm{E}[B_x^i] &\leq \sum_{b,c\geq 1} (1+\epsilon')b/c \cdot \Pr[B_x = b] \cdot \Pr[S_x = 1/c] \\ &= (1+\epsilon') \cdot \mathrm{E}[B_x] \cdot \mathrm{E}[S_x] \\ &\leq (1+\epsilon)(m \cdot dx) \cdot (1+\epsilon)/s\ , \end{aligned}$$

where ϵ and ϵ' can be made arbitrarily small. Hence,

$$\begin{aligned} \mathrm{E}[L_i] &= \int_{x=0}^{s\cdot d_i} \mathrm{E}[B_{g(i)+x}^i] \leq \int_{x=0}^{s\cdot d_i} (1+\epsilon'')m/s\ dx \\ &= (1+\epsilon'')m \cdot d_i\ , \end{aligned}$$

where $\epsilon'' > 0$ can be made arbitrarily small. In the same way it can be shown that $\mathrm{E}[L_i] \geq (1-\epsilon'')md_i$, where $\epsilon'' > 0$ can be made arbitrarily small. □

To complete the description of SHARE, we specify how SHARE adapts to a changing system. Suppose that there is a change in the capacities of the system from $d = (d_1, \ldots, d_n)$ to $d' = (d'_1, \ldots, d'_n)$. (We note that this also includes that a new bin enters the system, since this can simply be modelled by including it already in d with value 0.) In this case, SHARE performs a so-called *lazy update* strategy:

Let $0 < \lambda < 1$ be some fixed constant. λ specifies the *lazyness* of SHARE. SHARE only changes the capacity for bin i from d_i to d'_i if $d'_i \geq (1+\lambda)d_i$ or $d'_i \leq (1-\lambda)d_i$. This will cause the total capacities of the system to deviate from 1, but it will always be within $1 \pm \lambda$ and therefore will not endanger the results shown above as long as λ is sufficiently small. (That is, SHARE is still faithful with respect to the true capacity distribution.)

If the capacities of bins change, then balls will be moved in such a way that afterwards the call of UNIFORM$(b, C_{h(b)})$ results again in the correct position of the ball. There are various ways of solving this algorithmically, but we will only focus here on how many replacements of balls this would need, i.e. the competitive ratio of SHARE.

THEOREM 2.5. *If $s = \Omega(\ln N)$, then SHARE has a competitive ratio of at most $2+\epsilon$ for any $\epsilon > 0$.*

PROOF. Since SHARE is based on random hashing, it is easy to see that changes in the number of balls do not require SHARE to replace balls in order to remain faithful. Hence, we only have to consider changes in the capacities of the bins.

Suppose that we are given two capacity distributions, d and d', where d is the actual distribution used by the bins (that may not be identical with the distribution demanded by the system, since a lazy update rule is used) and d' is the new capacity distribution. Let α be chosen such that $\sum_i d_i = 1 + \alpha$. We know that $\alpha \in [-\lambda, \lambda]$. Furthermore, we know that in this case with $s = \Omega(\ln N)$, $\mathrm{E}[c_x] \in [(1-\epsilon)(1+\alpha)s, (1+\epsilon)(1+\alpha)s]$ for some $\epsilon > 0$ that can be made arbitrarily small. Thus, it holds for the expected share of bin i that $\mathrm{E}[S_i] \in [(1-\epsilon)(1+\alpha)d_i, (1+\epsilon)(1+\alpha)d_i]$.

Consider now some fixed bin i. If d'_i is within $(1 \pm \lambda)d_i$, then d_i does not change. Thus, the number of replacements caused by bin i is 0. Otherwise, $d'_i = (1+\beta)d_i$ for some β outside of $[-\lambda, \lambda]$. In this case, d_i will be set to d'_i. If $d'_i > d_i$, this causes in the worst case the replacement of an expected number of at most

$$\begin{aligned} &(1+\lambda)((1+\epsilon)d'_i m - (1-\epsilon)d_i m) \\ &\quad= (1+\lambda)(d'_i - d_i + \epsilon(d'_i + d_i))m \\ &\quad\leq (1+\lambda)(\beta d_i + \epsilon(2+\beta)d_i)m \\ &\quad\leq (1+\gamma)\beta d_i m = (1+\gamma)m \cdot |d'_i - d_i| \end{aligned}$$

balls for any $\gamma > 0$ if $\lambda > 0$ and $0 < \epsilon < \lambda \cdot \gamma/2$ are sufficiently small. This holds, because $\alpha \leq \lambda$ and $\beta \geq \lambda$. If $d'_i < d_i$, then we also get a bound of $(1+\gamma)m \cdot |d'_i - d_i|$ by just exchanging the positions of d_i and d'_i in the calculation above.

Hence, altogether SHARE only requires the replacement of at most $(1+\gamma)\sum_i |d'_i - d_i|m$ balls when changing from capacity distribution d to d'. Since the minimum amount of movements required for a perfectly faithful placement scheme is $\sum_{i:\, d_i > d'_i} (d_i - d'_i)m = \frac{1}{2}\sum_i |d'_i - d_i|m$, the theorem follows. □

To summarize the properties of SHARE: It is faithful, time- and space-efficient and amortized $(2+\epsilon)$-competitive for any $\epsilon > 0$. Its drawbacks are that the number of balls in a bin is not highly concentrated around the capacity (unless s is very large) and that its space complexity depends on N and not just on n. The next, more complicated scheme will remove these drawbacks.

3. THE SIEVE STRATEGY

Next we describe an alternative adaptive hashing strategy that does not have the drawbacks of the previous strategy and that does not rely on an extra placement strategy for uniform capacities. Some of the proofs of our results can be found in the appendix.

Our strategy requires hash functions that assign to each ball a real number chosen independently and uniformly at random out of the range $[0, 1)$. Suppose that initially the number of bins is equal to n. Let $n' = 2^{\lceil \log n \rceil + 1}$. We cut $[0, 1)$ into n' ranges of size $1/n'$ and we demand that every range is used by at most one bin. If range I has been assigned to bin i, then i is allowed to select any interval in

I that starts at the lower end of I. The intervals will be used in a way (described in more detail below) that any ball mapped to a point in that interval will be assigned to the bin owning it. We say that a range is *completely occupied* by a bin if its interval covers the whole range. A bin can own several ranges, but it is only allowed to own at most one range that is not completely occupied by it. Furthermore, we demand from every bin i that the total amount of the $[0,1)$ interval covered by its intervals is equal to $d_i/2$ (it will actually slightly deviate from that, but for now we assume it is $d_i/2$). This ensures the following property.

LEMMA 3.1. *For any capacity distribution it is possible to assign ranges to the bins in a one-to-one fashion so that each bin can select intervals in $[0,1)$ of total size equal to $d_i/2$.*

PROOF. Since every bin is allowed to have only one partly occupied range, at most n of the n' ranges will be partly occupied. The remaining $\geq n$ ranges cover a range of at least $1/2$, which is sufficient to accommodate all ranges that are completely occupied by the bins. □

So suppose we have an assignment of bins to intervals in their ranges such that the lemma is fulfilled. Then we propose the strategy described in Figure 2 to distribute the balls among the bins (the fall-back bin will be specified later). It is based on L random hash functions $h_1,\ldots,h_L : \{1,\ldots,M\} \rightarrow [0,1)$, where initially $L = \log n' + f$. The parameter f will be specified later. Figure 2 implies the following result:

Algorithm SIEVE(b)**:**
Input: number b of a ball
Output: bin number $\in \{1,\ldots,n\}$ that stores b
for $i = 1$ **to** L **do**
  set $x = h_i(b)$
  if x is in some interval of bin s **then return** s
return number of fall-back bin

Figure 2: The SIEVE algorithm.

THEOREM 3.2. *SIEVE can be implemented so that the position of a ball can be determined in expected time $O(1)$ using a space of $O(n)$ words (without considering the hash functions).*

PROOF. Since the bins occupy exactly half of the interval $[0,1)$, the probability that a ball succeeds to be placed in a round is $1/2$. Hence, the expected computation time of a ball position is $O(1)$.

The space requirement is $O(n)$ words, since information about the occupancy of $O(n)$ ranges has to be stored. □

Let a ball that has not been assigned to a bin in the for-loop of the algorithm above be called a *failed* ball. Obviously, the expected fraction of balls that fail is equal to $1/2^L$. Thus, the expected share of the balls any bin i (apart from the fall-back bin) will get is equal to $d_i(1-1/2^L)$. However, we want to ensure that every bin gets an expected share of d_i. To ensure this, we first specify how to select the fall-back bin.

Initially, the bin with the largest share is the fall-back bin. If it happens at some time step that the share of the largest bin exceeds the share of the fall-back bin by a factor of 2, then the role is passed on to that bin.

Next we ensure that every bin i gets an expected share of d_i. Let every non-fall-back bin choose an *adjusted share* of $d'_i = d_i/(1-1/2^L)$, and the fall-back-bin chooses an adjusted share of $d'_i = (d_i - 1/2^L)/(1-1/2^L)$. First of all, the adjusted shares still represent a valid share distribution, because

$$\sum d'_i = \frac{1-d_i}{1-1/2^L} + \frac{d_i - 1/2^L}{1-1/2^L} = 1 \, .$$

When using these adjusted shares for the selection of the intervals, now every non-fall-back bin i gets a true share of $(d_i/(1-1/2^L)) \cdot (1-1/2^L) = d_i$ and the fall-back bin gets a true share of $((d_i - 1/2^L)/(1-1/2^L)) \cdot (1-1/2^L) + 1/2^L = d_i$. Hence, the adjusted shares will ensure that the expected share of every bin is precisely equal to its capacity. Thus, we arrive at the following conclusion.

THEOREM 3.3. *SIEVE is perfectly faithful.*

In addition, a high concentration around the expected value can be shown.

THEOREM 3.4. *For every bin i, let the random variable L_i denote the number of balls placed in i. It holds for every $\epsilon > 0$:*

$$\Pr[L_i \geq (1+\epsilon)d_i m] \leq \mathrm{e}^{-\min[\epsilon^2,\epsilon]d_i m/3}$$

and

$$\Pr[L_i \leq (1-\epsilon)d_i m] \leq \mathrm{e}^{-\epsilon^2 d_i m/2} \, .$$

The theorem follows directly from the Chernoff bounds and our assumption that all balls choose their values in $[0,1)$ independently at random. Thus, $L_i = d_i m + O(\sqrt{d_i m \log n})$ w.h.p., which is a much better concentration around the expected value for L_i then achievable by SHARE with a reasonable amount of resources (see Lemma 2.3).

In order to show that SIEVE also has a very good adaptivity, we have to consider the following cases:

1. the capacities change
2. the number n' of ranges has to increase to accommodate new bins
3. the role of the fall-back bin has to change
4. the number L of levels has to increase to ensure that $1/2^L$ is below the share of the fall-back bin

As for the SHARE strategy, changes in the number of balls do not require SIEVE to replace balls in order to remain faithful, since SHARE is based on random hashing.

We begin with considering the situation that the capacities of the system change from $P = (p_1, p_2, \ldots)$ to $Q = (q_1, q_2, \ldots)$. Then we use the following strategy: every bin i with $q_i < p_i$ reduces its intervals in a way that afterwards it again partly occupies at most one range, and then every bin i with $q_i > p_i$ extends its share so that it also partly occupies afterwards at most one range.

It is easy to check that there will always be ranges available for those bins that increase their share so that every range is used by at most one bin. It remains to bound the expected fraction of the balls that have to be replaced.

LEMMA 3.5. *For any change from one capacity distribution to another that does not involve the change of the fall-back bin, the replacement strategy has a competitive ratio of 2.*

PROOF. In the following, let p_i' (resp. q_i') denote the adjusted share of bin i for P (resp. Q). For any i with $q_i < p_i$, the fraction of $[0,1)$ taken away from bin i is equal to $(q_i' - p_i')/2$. Furthermore, for any i with $q_i > p_i$, the fraction of $[0,1)$ added to bin i is equal to $(p_i' - q_i')/2$. Hence, the expected fraction of balls participating in the first placement round of SIEVE that are affected by the change in the distribution of shares is equal to $||P' - Q'||/2$, where

$$||P' - Q'|| = \sum_{i=1}^{n} |p_i' - q_i'| .$$

For the remaining balls that previously participated in the second round, the fraction affected by this is also equal to $||P' - Q'||/2$, and so on.

Now, for any $r \in \{1, \ldots, L\}$ let X_r denote the fraction of balls previously participating in round r and Y_r denote the fraction of these balls still participating in round r that have to be replaced. (We can exclude the failed balls, since any of these that still fail will be stored in the same fall-back bin.) In this case, $Y = \sum_{r=1}^{L} Y_r$ represents the total fraction of balls that need a replacement. Since for a given X_r, $\mathrm{E}_{X_r}[Y_r] \le \frac{1}{2}||P' - Q'||X_r$, we obtain for any given $X_1, \ldots, X_L$ that

$$\mathrm{E}_{X_1,\ldots,X_r}[Y] = \sum_{r=1}^{L} \mathrm{E}_{X_1,\ldots,X_r}[Y_r] \le \tfrac{1}{2}||P' - Q'|| \sum_{r=1}^{L} X_r .$$

We know that in each round the expected fraction of participating balls that is not placed is $1/2$. Hence, $\mathrm{E}[X_r] = 1/2^{r-1}$ for all r and therefore

$$\begin{aligned} \mathrm{E}[Y] &\le \tfrac{1}{2}||P' - Q'|| \sum_{r=1}^{L} \frac{1}{2^{r-1}} \\ &= \left(1 - \frac{1}{2^L}\right) ||P' - Q'|| \\ &= \left(1 - \frac{1}{2^L}\right) \frac{||P - Q||}{1 - 1/2^L} = ||P - Q|| . \end{aligned}$$

Since a perfectly faithful placement scheme would have to move at least a fraction of $||P-Q||/2$ of the balls, this proves the lemma. □

Next we consider the situation that the number n' of ranges has to increase. This happens if a new bin is introduced which requires $n' = 2^{\lceil \log n \rceil + 1}$ to grow. In this case, we simply subdivide each old range into two new ranges. Since afterwards the property is still kept that every bin partly occupies at most one range, nothing has to be replaced.

Consider now the situation that the role of the fall-back bin has to change. Recall that this happens if the bin with the maximum share has at least twice the share of the fall-back bin. Let s_1 be the old and s_2 be the new fall-back bin. Suppose that the number of bins in the system is n. Then s_2 has a share of at least $1/n$. At the time when s_1 was selected, the share of s_1 was at least as large as the share of s_2. Hence, the total amount of changes in the shares of s_1 and s_2 since then must have been at least $1/(2n)$. Changing from s_1 to s_2 involves the movement of an expected fraction of

$$\left|\frac{d_1 - 1/2^L}{1 - 1/2^L} - \frac{d_1}{1 - 1/2^L}\right| + \left|\frac{d_2}{1 - 1/2^L} - \frac{d_1 - 1/2^L}{1 - 1/2^L}\right|$$

of the balls, which is at most $\frac{3}{2^L - 1}$. If the f in the formula $L = \log n' + f$ is sufficiently large, then $\frac{3}{2^L-1} \ll \frac{1}{2n}$, and therefore the amount of work for the replacement can be "hidden" in the replacements necessary to react to changes in the capacity distribution of the system.

Next consider the situation that the number of levels L has to grow. Once in a while this is necessary, since for the case that many new bins are introduced the fall-back bin may not be able or willing to store a fraction of $1/2^L$ of the blocks. We ensure that this will never happen with the following strategy:

Whenever the share of the fall-back bin is less than $1/2^{L-t}$ for some integer t, we increase the number of levels from L to $L+1$.

This strategy will cause balls to be replaced. We will show, however, that also here the fraction of balls that have to be replaced can be "hidden" in the amount of balls that had to be replaced due to changes in the distribution requirement.

Let s_j be the fall-back bin that required an increase from $L-1$ to L (resp. the initial fall-back bin if no such bin exists), and let s_k be the current fall-back bin that requires now an increase from L to $L+1$. Then we know that the size of s must have been at least $1/2^{L-(t+1)}$ when it became a fall-back bin. Suppose that s_k took over the role of a fall-back bin from s_j. Then its share must have been twice as large then the share of s_j. Since its share was at most the share of s_j when s_j got the role as fall-back bin, the total amount of changes in the shares of s_j and s_k since then must have been at least $1/2^{L-t}$. This can also shown to be true for a longer history of fall-back bins from s_j to s_k. Changing from L to $L+1$ involves the movement of an expected fraction of at most

$$\left(\sum_{i \ne k}\left|\frac{d_i}{1 - 1/2^L} - \frac{d_i}{1 - 1/2^{L+1}}\right|\right) + \left|\frac{d_k - 1/2^L}{1 - 1/2^L} - \frac{d_k - 1/2^{L+1}}{1 - 1/2^{L+1}}\right|$$

of the balls, which is at most $\frac{2}{2^L-3}$. If t and $f \ge t$ are sufficiently large, then $\frac{2}{2^L-3} \ll 1/2^{L-t}$, and therefore also here the amount of work for the replacement can be "hidden" in the replacements necessary to accommodate changes in the distribution of shares.

Hence, we arrive at the following result.

THEOREM 3.6. *SIEVE is $(2+\epsilon)$-competitive, where $\epsilon > 0$ can be made arbitrarily small.*

Finally, we note that the SIEVE strategy can easily be executed in a distributed way: If every bin always knows the complete capacity distribution, then in the case of a change in the capacities, every bin can compute locally in a way consistent with the other bins, how the assignment of intervals to the ranges changes. Every bin can then check for itself whether there is a ball stored in it that has to be replaced and, if necessary, sends this ball to the correct bin. Similar strategies can be used for the other scenarios in which balls have to be replaced. Of course, this strategy does not only work for SIEVE but also for SHARE.

4. APPLICATIONS

In this section we list possible applications of our adaptive hashing schemes.

4.1 Distributed data servers

Consider the situation that we have a distributed data server or a storage area network. Such a system may have a large collection of disks, and it is quite likely that disks break down, are added, or have to be replaced. Previous data management strategies such as RAID have severe problems with these changes. In addition to being able to faithfully distribute data blocks among disks and achieving a high adaptivity, a dynamic placement scheme should also be able to ensure that requests to the data blocks have the same distribution as the data blocks themselves. Since both SHARE and SIEVE are based on random hash functions, it is easy to check that both the SHARE and the SIEVE strategy fulfill this property. In particular, the following theorem can be shown.

THEOREM 4.1. *SHARE can be set up so that the probability of a data request to be sent to disk i can be brought arbitrarily close to its demand d_i, and SIEVE even ensures that the probability of a data request to be sent to disk i is equal to its demand d_i.*

4.2 Web caching

A web cache – also called proxy server – is a network entity that satisfies HTTP requests on the behalf of a web server. In order to realize this, the web cache has its own disk storage and keeps in this storage copies of recently requested objects. Web caches are enjoying wide-scale deployment in the Internet for several reasons. First, a web cache can substantially reduce the response time for a client request, particularly if the bottleneck bandwidth between the client and the original server is much less than the bottleneck bandwidth between the client and the cache. Second, web caches can substantially reduce web traffic in the Internet. Already in 1998, over 75 percent of Internet traffic was web traffic, so a significant reduction in web traffic can translate to a significant improvement in Internet performance.

Multiple web caches, located at different places in the Internet, can cooperate to improve overall performance. An example of a cooperative caching system is the NLANR caching system, which consists of a number of backbone caches in the United States, and the Akamai caching system, which provides caching services for companies all over the world.

Clients can use a hash function to discover which cache stores an object. Consider now what happens when the set of active caching machines changes, or when each client is aware of a different set of caches. (Such situations are very plausible on the Internet.) In this case the clients may have an inconsistent view of the caches. Thus, we need a data placement scheme that is able to support inconsistent views. A *view* V is a demand distribution $(v_1, \ldots, v_n)$ that may deviate from the current demand distribution $D = (d_1, \ldots, d_n)$. The *consistency* of a view V is given by

$$\gamma_V = \sum_{i=1}^{n} \min[v_i, d_i] \ .$$

Obviously, $\gamma_V \in [0, 1]$ and the closer γ_V is to 1, the closer V is to D. To ensure that the consistency of a view is in close correlation with the probability of computing the correct bin for a ball, we need one more definition.

A placement scheme is called *oblivious* if no matter how a system evolved to reach a demand distribution D, the distribution of balls among the bins will be the same. In a non-oblivious scheme, it may happen that a client cannot determine the correct position of any of the balls although its view matches the current demand distribution. Hence, it is important to use oblivious placement schemes to ensure this does not happen. SHARE together with NEAREST NEIGHBOR is oblivious, whereas SIEVE is not. Hence, we will only consider SHARE in combination with NEAREST NEIGHBOR. We can show the following result that generalizes a result by Karger et al. [4] from uniform to non-uniform demand distributions.

THEOREM 4.2. *Suppose we use SHARE in combination with NEAREST NEIGHBOR. For any constant $0 < \gamma < 1$, $s = \Theta(\log N)$ and $\delta = \Theta(1/\log^2 N)$ can be chosen so that for any view V with consistency at least γ it holds: If every data element has $\gamma^{-1} \cdot \Theta(\log N)$ copies, then w.h.p. at least one location of a copy is the same for both V and the current demand distribution D.*

PROOF. We prove the theorem in two steps. First, we consider some fixed copy b' of a ball b and show that the probability that the bin in which b' is stored is not the same for V and D is some constant smaller than 1. Then, we show that the probabilities of different copies of b to run into such a situation are nearly independent, so that the probability that for all copies b' the location of b' is not the same for V and D is polynomially small. This would result in the theorem.

So consider now a fixed copy b' of a ball b and let $x = h(b')$. Let C_x be the set of bins with intervals containing x w.r.t. D and C'_x be the set of bins with intervals containing x w.r.t. V. Furthermore, let $O_x = C_x \cap C'_x$. For every constant $\epsilon > 0$ we can choose an $s = \Theta(\log N)$ so that $|C_x|$ and $|C'_x|$ are within $(1 \pm \epsilon)s$ w.h.p. and $|O_x|$ is within $(1 \pm \epsilon)\gamma s$ w.h.p. To simplify the following calculations, we will assume that $|C_x| = |C'_x| = s$ and $|O_x| = \gamma s$.

Clearly, the probability that the location of b' is the same for D and V is equal to the probability that b' is placed in a bin in O_x when using NEAREST NEIGHBOR on the bin set $C_x \cup C'_x$. Hence,

$$\begin{aligned} &\Pr[\text{bin of } b' \text{ the same for } D \text{ and } V] \\ &= \frac{|O_x|}{|C_x \cup C'_x|} \geq \frac{\gamma s}{2s} = \frac{\gamma}{2} \ . \end{aligned}$$

Thus, the probability that the bin of b' is not the same for D and V is at most $1 - \gamma/2$.

Now we want to determine the probability that all of the copies $b'_1, \ldots, b'_k$ of a ball b have bins that are not the same for D and V. Here, we face the problem that there are dependencies among these probabilities: If it is known that some copy b'_i is not in the same bin for D and V and for some other copy b'_j, $h(b'_j)$ and $h(b'_i)$ are so close that they may have the same set of bins (which may happen for $|h(b'_j)-h(b'_i)| \leq \delta s$), then b'_j may also not be in the same bin for D and V. Thus, we would not get an extra probability for b'_j. Nevertheless, the following lemma can be shown. In this lemma, A_i denotes the event that b'_i is not in the same bin for D and V.

LEMMA 4.3. *If $\delta \leq \gamma/(4k \cdot s)$, then it holds for every $i \in \{1, \ldots, k-1\}$:*

$$\Pr[A_{i+1} \mid A_1 \wedge \ldots \wedge A_i] \leq 1 - \gamma/2 + i \cdot 2\delta s \ .$$

PROOF. Let $x_i = h(b'_i)$ for all $i \in \{1, \ldots, k\}$. Furthermore, let R_i be the event that x_i is within a radius of δs of some x_j, $j < i$. Then it holds that

$$\begin{aligned} &\Pr[A_{i+1} \mid A_1 \wedge \ldots \wedge A_i] \\ &= \Pr[R_{i+1}] \cdot \Pr[A_{i+1} \mid R_{i+1} \wedge A_1 \wedge \ldots \wedge A_i] \\ &\quad + \Pr[\bar{R}_{i+1}] \cdot \Pr[A_{i+1} \mid \bar{R}_{i+1} \wedge A_1 \wedge \ldots \wedge A_i] \\ &\leq \Pr[R_{i+1}] + \Pr[A_{i+1} \mid \bar{R}_{i+1} \wedge A_1 \wedge \ldots \wedge A_i] \ . \end{aligned}$$

Obviously, $\Pr[R_{i+1}] \leq i \cdot 2\delta s$. Hence, it remains to show that $\Pr[A_{i+1} \mid \bar{R}_{i+1} \wedge A_1 \wedge \ldots \wedge A_i] \leq 1-\gamma/2$ (up to some negligible ϵ terms that we do not consider). Since $|C_x| = |C'_x| = s$ for all $x \in [0, 1)$ (up to negligible ϵ terms) and none of the x outside of the δs radius of the x_j with $j \leq i$ can have an interval in C_x that also contains such a point x_j, the probability for A_{i+1} to be true at any of these points x is independent of the $A_1, \ldots, A_i$. Hence,

$$\Pr[A_{i+1} \mid \bar{R}_{i+1} \wedge A_1 \wedge \ldots \wedge A_i] \leq 1 - \gamma/2 \ ,$$

which completes the proof. □

From Lemma 4.3 it follows with $\delta \leq \gamma/(8k \cdot s)$ that

$$\Pr[A_1 \wedge \ldots \wedge A_k] \leq (1-\gamma/4)^k \leq \mathrm{e}^{-\gamma k/4} \ .$$

Thus, choosing $k = \Theta(\gamma^{-1} \log N)$, the probability that no copy of ball b is in the same bin for D and V can be made polynomially small, which completes the proof.

4.3 Peer-to-peer networks

Peer-to-peer (or P2P) networks are overlay-networks which connect their nodes via some existing network and work together to provide distributed services such as search, content integration and administration. Two main properties characterize such a network. There is not central node that handles the communication between peers like in a client-server model. Instead, queries fan out over the network, and results are collected and propagated back to the originating node. Furthermore, the topology of the P2P is constantly changing, since nodes may join and leave the network at any time. Popular examples of P2P protocols are Gnutella, Limewire, Bearshare, and Kazaa.

Several challenging problems have to be solved to ensure that

1. the topology of a peer-to-peer network always remains connected,
2. the nodes are given degrees so that the full potential of the peer-to-peer network can be exploited, and
3. how to rapidly adapt to changes in the number or characteristics of the peers.

The last two items require the use of adaptive hashing techniques for non-uniform demands, since different peers may have different computational power or different connections to the Internet (dial-up modems or DSL). In order to solve all three problems, we propose a combination of an approach by Pandurangan et al. [5] and our adaptive hashing techniques. Pandurangan et al. presented a mechanism to integrate nodes into the network that ensures that the peer-to-peer network is always connected, has a constant degree, and a diameter of $O(\log n)$, w.h.p. In addition, the server that will introduce new nodes to the network always only has to hold a constant number of nodes at any time. Their approach is optimal in many ways, but it does not allow to take into account the non-uniform characteristics of peers, and no proof is given that their approach creates an expander between the nodes. We solve these problems by using SIEVE to establish an additional topology on top of the Pandurangan et al. network. This is done as follows:

Suppose that the current demand distribution (reflecting the power of the peers) is $d = (d_1, \ldots, d_n)$ and let $f(n)$ be a function that determines the total number of undirected edges that should be used to connect the nodes. Suppose, for example, that $f(n) = c \cdot n$ for some constant $c \in \mathbb{N}$. Then each node is considered as the endpoint of c edges, and the other endpoints of the edges are represented as balls that are distributed among the peers as prescribed by SIEVE. This results in the following theorem.

THEOREM 4.4. *Suppose that $c \geq 3$. Then the P2P strategy above is faithful (concerning the degree of a node) up to an additive constant and amortized $(1+\epsilon)$-competitive. Furthermore, if the demand distribution is uniform, the network formed by SIEVE is an expander, w.h.p.*

PROOF. The faithfulness and competitiveness are obvious. Thus, it remains that the network formed by SIEVE in the uniform case is an expander, w.h.p. Since SIEVE ensures that in a uniform demand distribution the edges choose their endpoints uniformly and independently at random, standard techniques that show that a random graph is an expander w.h.p. also imply that SIEVE creates an expander w.h.p. To give an idea, it just has to be shown for every $s \leq n/c$ that for every set U with $|U| = s$, the probability that the neighborhood of U is less than s is polynomially smaller than $1/\binom{n}{s}$, the number of all possible subsets of nodes of size s. This would imply that w.h.p. no subset U of size at most n/c has an expansion that is below $|U|$, implying that the graph is an expander. □

5. CONCLUSION

In this paper we presented two compact, adaptive placement schemes for non-uniform demands. Several open problems remain, since these schemes are not optimal. For example, is it possible to construct a compact scheme that is (almost) perfectly faithful and (close to) 1-competitive and requires an amount of space that only depends on the current number of bins n (and maybe the current number of balls in a logarithmic way)? SHARE also depends on N, and SIEVE depends on the maximum number of bins that have been in the system so far. Also, both are not 1-competitive but only $2 + \epsilon$-competitive for any $\epsilon > 0$.

Another question that might be worth to investigate is how to further reduce the deviation of the number of balls in a bin from its expected value beyond what SIEVE can achieve.

6. ACKNOWLEDGEMENTS

We would like to thank Friedhelm Meyer auf der Heide for many stimulating discussions.

7. REFERENCES

[1] 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.

[2] 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.

[3] T. Cortes and J. Labarta. Extending heterogeneity to RAID level 5. In *USENIX 2001*, Boston, June 2001.

[4] 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.

[5] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *Proc. of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 492–499, 2001.

[6] 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.

[7] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the ACM SIGCOMM 2001*, pages 149–160, Aug. 2001.

[8] R. Zimmermann and S. Ghandeharizadeh. HERA: Heterogeneous extension of RAID. In *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, 2000.