Weighted Distributed Hash Tables

Christian Schindelhauer*
Heinz Nixdorf Institute and
Computer Science Department
University of Paderborn, Germany
schindel@uni-paderborn.de

Gunnar Schomaker*
Heinz Nixdorf Institute and
Computer Science Department
University of Paderborn, Germany
pinsel@uni-paderborn.de

ABSTRACT

We present two methods for weighted consistent hashing also known as weighted distributed hash tables. The first method, called *Linear Method*, combines the standard consistent hasing introduced by Karger et al. [9] with a linear weighted distance measure. By using node copies and different partitions of the hash space, the balance of this scheme approximates the fair weight relationship with high probability. The second method, called the *Logarithmic Method*, uses a logarithmic weighted distance between the peers and the data to find the corresponding node. For distributing one data element it provides perfect weighted balance. To provide this distribution for many data elements we use partitions to achieve a fair balance with high probability. These methods provide small fragmentation, which means that the hash space is divided into at most $\mathcal{O}(n \log n)$ intervals. Furthermore, there is an efficient data structure that assigns data elements to the nodes in expected time $\mathcal{O}(\log n)$. If small fragmentation is not an issue one can replace the use of partitions by a method we call double hash functions. This method needs $\mathcal{O}(\mathrm{n})$ for assigning elements to a node, yet it can be directly used for Storage Area Networks, where the number of nodes is small compared to participating nodes in Peer-to-Peer networks.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems, Distributed Databases; E.1 [**Data Structures**]: Distributed data Structures; E.2 [**Data Storage Representations**]: Hash-table representations; G.3 [**Probability and Statistics**]: Probabilistic algorithms

General Terms

Algorithms, Theory

Keywords

Peer-to-Peer Networks, Storage Area Networks, consistent hashing, adaptive hashing, web caching, non-uniform disks

1. INTRODUCTION

Karger et al. [9] introduced the notion of Consistent Hashing, aka. Distributed Hash Table. Such a scheme gives a mapping from a set of data elements to a dynamic set of hosts. In the original setting data was to be distributed among different sets of hosts, so called views. The goal was to avoid swamped servers, by decreasing the usage of memory and to balance data fairly among the views.

For this, the hosts are mapped to some range using a hash function. Also the data elements are mapped to the very same range using an appropriate hash function. Now in every view (relevant sub-set of hosts) a data element is stored on a host, if this host's image in the hash range minimizes the distance to the image of the data element. If a sufficient number of copies of the hosts are mapped to the range, one can show that this leads to a fair balance of data elements [14, 6]. Now, if a new host is added to this system, then only data elements need to be reassigned which will be stored on the new host. This feature is called consistency.

Such distributed hash tables are universally applicable to many areas of distributed computing. First they are introduced to distribute web sites among servers distributed around the globe relieving hot spots in the Internet [9]. Besides the area of Web Caching they are popular in Peer-to-Peer Networks, see CAN [15], Chord [16], Pastry [7], Tapestry [8], and many more. A further important application field are Storage Area Networks (SAN), to overcome problems induced by huge RAID arrays. Here, the task is to distribute data on multiple heterogeneous disks that act like one virtual disk [3].

In this paper we consider the more general case of weighted consistent hashing and compared to the original approach every host v_i comes now with a positive weight w_i. Let $W = \sum_{i \in V} w_i$ be the overall weight. Then, the goal is to distribute a data element with probability w_i/W to node v_i. Clearly, such a weighting is a helpful extensions to all the above named application areas. Especially for Peer-to-Peer networks and Storage Area Networks such an extension is crucial. In Peer-to-Peer networks participants are not necessarily equipped with equal storage devices or transmission

*This work was partially supported by the DFG Sonderforschungsbereich 376 and by the EU within 6th Framework Programme under contract 001907 Dynamically Evolving, Large Scale Information Systems (DELIS).

SPAA'05, July 18–20, 2005, Las Vegas, Nevada, USA.

bandwidth. Hence, a weighting can improve performance. Maybe, in SANs the need for such a weighting is even higher. The capacity of disks varies enormously, and a fair distribution of data is crucial for the performance [3].

The naive approach for introducing a weighted version of consistent hashing is to use $\left\lceil \frac{w_i}{\min_{j \in V}\{w_j\}} \right\rceil$ copies for each peer w_i. This is not feasible, if $\max_{j \in V}\{w_j\}/\min_{j \in V}\{w_j\}$ is too large. Furthermore, interesting nodes with small weights increases the number of copies of all nodes.

Brinkmann et al. [5] presented a scheme to overcome this problem, but not as elegant as the original weighted consistent hashing and still uses a large number of copies. Furthermore, small disks are under-utilized and the scheme has to undergo special reorganization procedures if too many disks are in- or excluded.

We present two elegant and intuitive methods, called the Linear and the Logarithmic Method, that overcome these problems. In a nutshell these schemes work as follows. Both schemes map data elements and nodes to a range using hash functions. In the *Linear Method* for each data element x a height $H_i = d_i(x)/w_i$ is computed where $d_i(x)$ denotes the distance of the hash value of x to the hash value of host i in the hash range of the images of the data element and the nodes v_i. Now x is assigned to the host that minimizes this height H_i. The Logarithmic Method is essentially the same, yet using the height $H_i = -(\ln(1 - d_i(x))/w_i$ for finding the host v_i for the data element minimizing this term.

The Linear Method comes historically first and is already mentioned and implemented in [2]. We know that the Logarithmic Method outperforms the Linear Method. Yet, knowing the Linear Method helps to understand the Logarithmic Method. Therefore and because of its simplicity and elegance we think the Linear Method is worthwhile to be presented here.

These two methods are very close to the original setting of consistent hashing. Therefore, they are applicable to many areas where distributed hash tables are in use. In this paper we will present the essential features and limitations of these schemes. The main measures are fairness, quality of the weighting, and fragmentation which measures the number intervals in the hash range.

The paper is organized as follows. In the following section we present related research. Then, we present the basic concept of consistent hashing. In Section 4 we present the Linear Method, how the use of copies and partitions improves its performance, and present an efficient data structure for finding the host of a data element. At the end of the section we show the limitations of the Linear Method. In Section 5 we present the Logarithmic Method improving the Linear Method. We show how partitions improve this scheme and present an efficient data structure. In Section 6 we discuss further techniques helpful for these weighted distributed hash tables. This is the use of the so-called double hash function for Storage Area Networks. We show how the volatility of data elements can be determined and present isomorphic weighted hashing schemes. In the last section we summarize our results.

2. RELATED RESEARCH

Distributed hash tables are commonly based on consistent hashing [9] scheme or on the work of Plaxton et al. [13]. Usual distributed hash tables includes CAN[15], Chord[16], Distance Halving[12], Koorde[10], Pastry[7] or Tapestry [17] to name some of them. All these approaches have in common that they are for homogeneous purpose only, which means their methods neglects somehow the heterogeneity of nodes.

We have deployed a consistent hashing scheme respecting the heterogeneous properties of nodes. These nodes might represent storage devices in a SAN or peers in a P2P Network and therefor the scheme can be deployed as placement function for data elements, where weights are used to assign node responsibilities within a hash range.

In [2] such a weighted consistent hashing scheme was mentioned and presented in a poster session. Actually they have used the Linear Method based on the research of the authors of this paper. In their Peer-to-Peer approach they used the weighted consistent hashing scheme to distribute route lookup information within hierarchical clusters. This hierarchy was deduced from a landmark system, that identifies responsible nodes to determine the next hop for route completion.

Our association of hash values to hash locations works with a concept that is similarly to the scheme introduced by [4], yet improves on the balancing properties and can be evaluated more efficiently by this distributed network. In [5] Brinkmann et al. the authors introduced several criteria a placement scheme needs to fulfill, like a faithful distribution, efficient localization, and fast adaptation, this seems to weaken the orignal notion of Karger [9]. One will see that both schemes presented here accomplish the original criteria of consistency, that induces a fair distribution. An additional benefit of our scheme is that it still works correct and fair with a few number of nodes. Compared to SHARE [5] there is also no need to define a stretch factor s for covering the hash interval. In practice this implies to estimate this factor, which hardly depends on n, and is needed to achieve coverage [3]. In our weighted consistent hashing approach there is coverage guaranteed. As well there is no need to restrict the capacity of nodes by including virtual bins. Furthermore there is no need to know all node capacities in the *Logarithmic Method*, because of its self-scaling property when nodes are inserted or removed. By means, one can directly use the unscaled weights, e.g. megabytes for storage devices. The second strategy introduced in [5] called SIEVE takes use of a fallback bin to ensure data assignment. This bin is typically the node with largest capacity and is needed in addition to assure a balanced behavior. Such a construct is also not needed in our approach.

Eventually, we introduce a relevant, simple, and elegant weighted consistent hashing scheme including two basic assignment concepts, which both are easy to implement and also compatible to many areas of application, like Peer-to-Peer networks, Storage Area Networks or other related research fields, where the heterogeneity of nodes is the tremendous barrier that has to be conquered. Furthermore, this model reduces the hash interval fragmentation, and deploys smoothly fading techniques for appearing or leaving nodes, and enables the involvement of migration prediction for assigned data elements, which to our knowledge has not been investigated until now, even for the homogeneous case.

3. CONSISTENT HASHING

Given a set of n nodes $V = \{v_1, \ldots, v_n\}$ (aka. hosts, peers) with weights $w_i := w(v_i) \in \mathbb{R}^+$ and data elements $X = \{x_1, \ldots, x_m\}$. The weighted consistent hashing maps

all elements $f_V : X \to V$. Let $A_V(x) := f_V^{-1}(v)$ denote the data elements assigned to v and $M_V(v) := |A_V(v)|$ the number of elements in v.

DEFINITION 1. *A mapping is called* **consistent**, *if for* $V \subset V'$ *all* $v \in V$*:*

$$A_{V'}(v) \subseteq A_V(v) \ .$$

The goal is to assign each node $v \in V$ a fair share according to its relative weight, i.e. for m data elements

$$M_V(v) = m \cdot \frac{w_v}{\sum_{i \in V} w_i} \ .$$

Consistent Hashing, as introduced in [9], uses two-sided hashing into some continuous range $[0, 1)$. First the nodes are randomly mapped into this set using a hash function $h_1 : V \to [0, 1)$. Then, the data elements are mapped using a hash function $h_2 : X \to [0, 1)$. We may assume that appropriate hash functions are used that behave like independently distributed random variables. In the original hash functions data elements are assigned to the node which is closest to the element in $[0, 1)$.

We do not discuss the issue of selecting an appropriate hash function. However, the approach of [9] suffers under the coupon collector problem. So, there are nodes receiving intervals in the hash range that are up to a factor of $\mathcal{O}(\log n)$ times higher than the average size. This problem can be easily handled by using some $\mathcal{O}(\log n)$ copies of each node. Then by applying Chernoff bounds one can show that the error reduces to a small factor of $1 \pm \epsilon$ for any $\epsilon > 0$. Recently, other approaches have been presented for this problem [11]. It is an open problem how the principle of multiple choice, or the principle of two choices can be applied to this weighted distributed hashing.

4. THE LINEAR METHOD

In this section we present the *Linear Method* in detail. The method, as mentioned, is applicable for data placements in the area of SAN. In advance to improve clarity we now consider an example containing five heterogeneous storage devices $\{v_1, \ldots, v_5\}$ with different capacities $c(v_i) = w_i$, e.g. $w_1 = 2, w_2 = 5, w_3 = 1, w_4 = 0.8, w_5 = 6$. To place the disks into the range $[0, 1)$ we use an appropriate hash-function $h(v_i)$. Doing so for the first four devices may lead to the following positions $s_1 = 0.5, s_2 = 0.8, s_3 = 0.35, s_4 = 0.1$. If we examine H_i with respect to the minimum one can easily see how the mapping range gets divided and which fragments belong to which node, see Fig. 1. Similar to Karger [9], the location of a data element is determined by mapping it into the range too. The resulting position identifies the associated device, where the element belongs to.

Another crucial problem in this area is the scalability of such systems, by means devices might be removed or added. In our example we add a further device v_5 at the randomly chosen position $s_5 = 0.2$. This results to the following fragmentation points [0, 0.1, 0.157, 0.2, 0.35, 0.38, 0.5, 0.65, 0.8, 1), and leads to shares obtained from the Figure 1 for $v_1 : 15\%, v_2 : 34,3\%, v_3 : 3\%, v_4 : 5,7\%, v_5 : 42\%$, whereas the expected shares were $v_1 : 13,51\%, v_2 : 33,78\%, v_3 : 6,76\%, v_4 : 5,41\%, v_5 : 40,54\%$. One can observe that some x_i previously assigned to v_1 are now assigned to the new device and that this insertion has an impact on other assignments too. To preserve placement consistency the already

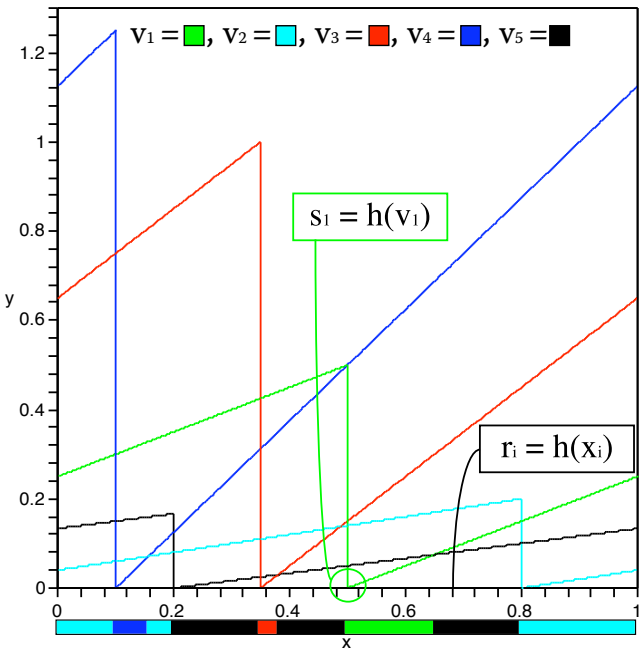

Figure 1: The Linear Method

assigned elements must be moved to the new device, all the other elements stay untouched. If a device is removed and the remaining capacity is sufficient enough to compensate the re-movement the procedure is analogous. To level out such re-movements other strategies are possible too.

The results in following subsections show that this scheme provides a fairly balanced distribution for data elements and is also capable to compensate dynamics with an minimal effort, concerning data movement or system reorganization.

4.1 The basic method

A set of nodes $V = \{v_1, \ldots, v_n\}$ and a set of data elements $X = \{x_1, \ldots, x_m\}$ are mapped to the continuous set $M = [0, 1)$ using appropriate hash functions that behave like uniform independent random variables. Each node v_i is associated with a capacity $w_i \in \mathbb{R}^+$. The hash function* h maps each data element x_i into M denoted by $r_i = h(x_i)$. Similarly we receive $s_i = h(v_i)$ denoting the position of node v_i in the set M.

We define the scaled distance function as

$$D_w(r, s) := \frac{((s - r) \bmod 1)}{w} \ ,$$

where $a \bmod 1 := a - \lfloor a \rfloor$, i.e. $D_w(r, s) := \frac{1}{w}((s-r) - \lfloor s-r \rfloor)$. According to the *Linear Method* we assign each data item r_i to the node v_i which minimizes the term $D_{w_v}(r_i, s_v)$. For a position $r \in [0, 1)$ we also call this term $H(r) := \min_{v \in V} D_{w_v}(r, s_v)$ the height of r.

This Linear Method is consistent: If the capacity of a single node increases or if a node is inserted, then only data items need to be reassigned which will be associated to this node. No other data elements will be reassigned. If a node disappears or its capacity decreases then only data elements from this node are reassigned to other nodes (according to

*For simplicity we name the hash function h for nodes and data elements mapped to M.

the above weighting). Again, no other data elements will be reassigned.

LEMMA 1. *Given n nodes with weights $w_1, \ldots, w_n$. Then the height $H(r)$ assigned to a position r in M is distributed as follows:*

$$P[H(r) > h] = \begin{cases} \prod_{i\in[n]}(1 - hw_i) , & \text{if } h \le \min_i\{\frac{1}{w_i}\} \\ 0 , & \text{else} \end{cases}$$

THEOREM 1. *The Linear Method stores with probability of at most $\frac{w_i}{W-w_i}$ a data element at a node v_i, where $W := \sum_{i=1}^{|V|} w_i$.*

PROOF. Let R denote the position where this data element is inserted. Let H_i denote the height of the data element. Note that all these random variables are independent and uniformly distributed (since we consider only one data element). Hence, the probability that s_i is at most in weighted distance h (or the element receives at most height h from node v_i) is described as follows.

$$\mathrm{P}[H_i \le h] = \begin{cases} 1 , & h \ge \frac{1}{w_i} \\ h \cdot w_i & \text{else.} \end{cases}$$

From this we can determine the probability, that an element receives a height in the interval $[h, h+\delta]$ for node v_i and receives greater heights for all other nodes. Let $h + \delta \le \frac{1}{w_i}$:

$$\mathrm{P}[H_i \in [h, h+\delta] \wedge \forall j \ne i : \ H_j > h] =$$

$$\begin{cases} 0 , & \exists j \ : \ h \ge \frac{1}{w_j} \\ \delta w_i \prod_{j\ne i}(1 - hw_j) & \text{else} . \end{cases}$$

Let $P_{i,h,\delta} := \delta w_i \prod_{j\ne i}(1 - hw_j)$. Now, an upper bound on the probability that an element is assigned to node v_i is given by the sum $\sum_{m=1}^{\infty} P_{i,\delta m,\delta}$. Note that for $h = m\delta$

$$P_{i,h,\delta} = \delta w_i \prod_{j\ne i}(1 - hw_j) \le \delta w_i e^{-\delta m \sum_{j\ne i} w_j} .$$

Now let $a := \sum_{j\ne i} w_i > 0$, then the sum $\sum_{m=1}^{\infty} P_{i,\delta m,\delta}$ can be transformed into an integral if δ tends to 0:

$$\begin{aligned} \lim_{\delta\to 0} \sum_{m=1}^{\infty} P_{i,\delta m,\delta} &\le \lim_{\delta\to 0} \sum_{m=1}^{\infty} w_i \delta e^{-a\delta m} \\ &= \int_{x=0}^{\infty} w_i e^{-ax} dx = \frac{w_i}{a} \\ &= \frac{w_i}{\sum_{j\ne i} w_j} \end{aligned}$$

□

4.2 The Use of Copies

Each node participates in the following scheme with $\lceil \frac{2}{\epsilon} + 1 \rceil$ copies for some $\epsilon > 0$. We do not formalize this property and treat each copy separately. We simply note that the number of nodes is increased by this constant factor. We need this to ensure the following inequality for all $j \in \{1, \ldots, n\}$:

$$W := \sum_{i=1}^{n} w_i \ \le \ \frac{1}{1 - \frac{1}{2}\epsilon} \sum_{i\ne j} w_i \ \le \ (1+\epsilon) \sum_{i\ne j} w_i .$$

THEOREM 2. *Let $\epsilon > 0$. Then, the Linear Method using $\lceil \frac{2}{\epsilon} + 1 \rceil$ copies assigns one data element to node v_i with probability p_i where*

$$(1 - \sqrt{\epsilon}) \cdot \frac{w_i}{W} \ \le \ p_i \ \le \ (1+\epsilon) \cdot \frac{w_i}{W} .$$

PROOF. We will now prove the first part. We use Theorem 1 and conclude

$$\frac{w_i}{\sum_{j\ne i} w_j} \ \le \ \frac{1}{1 - \frac{1}{2}\epsilon} \frac{w_i}{\sum_j w_j} \ \le \ (1+\epsilon)\frac{w_i}{W} .$$

For the lower bound we use the following lemma.

LEMMA 2. *For $\epsilon' > 0$, $h \le \frac{\epsilon'}{\max_j\{w_j\}}$, for all $i \in [n]$:*

$$e^{-hW/(1-\epsilon')} \ \le \ \prod_j (1 - hw_i) \ \le \ \prod_{j\ne i} (1 - hw_i) .$$

PROOF. Note that for $k > 1$: $(1 - \frac{1}{k})^{k-1} > \frac{1}{e}$. Then for $x \in (0, 1)$:

$$1 - x > (1 - x)^{\frac{-x}{1-x}}$$

Therefore for $h \le w_i$:

$$(1 - hw_i) > e^{-\frac{hw_i}{1-hw_i}}$$

Note that $hw_j \le \epsilon'$

$$\begin{aligned} \prod_j (1 - hw_j) &\ge e^{-\sum_{j\ne i} \frac{hw_j}{1-hw_j}} \\ &\ge e^{-\sum_j \frac{hw_j}{1-\epsilon'}} \\ &\ge e^{-hW/(1-\epsilon')} . \end{aligned}$$

□

A sufficient condition that a data element is assigned to node v_i is the following:

$$\mathrm{P}[H_i \in [h-\delta, h] \wedge \forall j \ne i : \ H_j > h] =$$

$$\begin{cases} 0 , & \exists j \ : \ h \ge \frac{1}{w_j} \\ \delta w_i \prod_{j\ne i}(1 - hw_j) & \text{else} . \end{cases}$$

let $P'_{i,h,\delta} := \delta w_i \prod_{j\ne i}(1 - hw_j)$ for $h \le \min\{1/w_j\}$. Now, the sum

$$S = \lim_{\delta\to 0} \sum_{m=1}^{\frac{\epsilon'}{\delta \max\{w_i\}}} P'_{i,\delta m,\delta} = \int_{h=0}^{\frac{\epsilon'}{\max\{w_i\}}} w_i \prod_{j\ne i}(1 - hw_j) dh$$

gives an lower bound on the probability that an element is assigned to node v_i. From Lemma 1 it follows for $\epsilon = \epsilon'^2$:

$$\begin{aligned} S &> \int_{h=0}^{\frac{\epsilon'}{\max\{w_i\}}} w_i e^{-hW/(1-\epsilon')} dh \\ &= (1-\epsilon')\frac{w_i}{W}\left(1 - e^{-\frac{\epsilon' W}{(1-\epsilon')\max\{w_i\}}}\right) \\ &\ge (1-\epsilon')\frac{w_i}{W}\left(1 - e^{-\frac{\epsilon'}{(1-\epsilon')\epsilon}}\right) \\ &\ge (1-\epsilon')\left(1 - \frac{(1-\epsilon')\epsilon}{\epsilon'}\right)\frac{w_i}{W} \\ &\ge (1-\epsilon')\left(1 - \frac{\epsilon}{\epsilon'}\right)\frac{w_i}{W} \\ &\ge (1-\sqrt{\epsilon})\frac{w_i}{W} \end{aligned}$$

```
Insert-Node(v, T)
begin
  s_v ← h(v)
  T* ← Insert-Table(T*, s_v, v)
  ℓ ← ⌊log_2 w_v⌋
  T_ℓ ← Insert-Table(T_ℓ, s_v, v)
  return (T)
end.
```

Figure 2: Insert a node v

4.3 The Use of Partitions

Note that after nodes have been assigned to the hash range then the probabilities p_i remain the same because they correspond to the intervals assigned to the node. Therefore the previous result does not imply that every node receives data elements according to this probability. Even in the balanced case the coupon collector problem occurs such that some nodes receive intervals which are a factor of $\mathcal{O}(\log n)$ larger than desired.

One can try solve this by introducing more copies. We suggest to partition the hash range into $\mathcal{O}(\log n)$ partial intervals, called partitions, and apply the Linear Method to each of these intervals.

The nodes $V = \{v_1, \ldots, v_n\}$ are mapped to each of the continuous set $M_1, \ldots, M_k$ with $M_i = [(i-1)/k, i/k)$ (with some copies) and the data elements $X = \{x_1, \ldots, x_m\}$ are mapped the whole interval $[0, 1)$ using hash functions.

THEOREM 3. *For all $\epsilon, \epsilon' > 0$ and $c > 0$ there exists $c' > 0$ such that when we apply the Linear Method to n nodes using $\lceil \frac{2}{\epsilon} + 1 \rceil$ copies and $c' \log n$ partitions, the following holds with high probability, i.e. $1 - n^{-c}$.*

Every node $v_i \in V$ receives all data elements with probability p_i such that

$$(1 - \sqrt{\epsilon} - \epsilon') \cdot \frac{w_i}{W} \;\leq\; p_i \;\leq\; (1 + \epsilon + \epsilon') \cdot \frac{w_i}{W} \;.$$

PROOF. This result follows by Theorem 2 and applying Chernoff bounds. □

```
Insert-Table(T, s, v)
begin
  N ← size(T)
  S ← Σ_{i=0}^{N-1} |T[i]|
  if T is empty then
    N ← 1
    T[0] ← {v}
  else if N ≤ S + 1 then
    N ← 2N
    T'[0, ..., N − 1] ← ∅
    for all v ∈ ⋃_{i=0}^{N/2−1} T[i] ∪ {v} do
      T'[⌊s/N⌋] ← T'[⌊s/N⌋] ∪ {v}
    od
    T ← T'
  else
    T[⌊s/N⌋] ← T[⌊s/N⌋] ∪ {v}
  fi
  return(T)
end.
```

Figure 3: Insert a node v into a table

```
Delete-Node(v, T)
begin
  s_v ← h(v)
  T* ← Delete-Table(T*, s_v, v)
  ℓ ← ⌊log_2 w_v⌋
  T_ℓ ← Delete-Table(T_ℓ, s_v, v)
  return (T)
end.
```

Figure 4: Delete a node v

4.4 Efficient Data Structure

THEOREM 4. *There is an algorithm that determines for a data element the corresponding node according the Linear Method in expected time $\mathcal{O}(\log n)$. The data structure has size $\mathcal{O}(n)$. Inserting and deleting nodes in this data structures needs amortized time $\mathcal{O}(1)$.*

PROOF. We present the algorithms for the data structure for the plain Linear Method without copies and partitions, see Fig. 6, 2, 4, 3, and 5.

The nodes are classified according their weights into the sets $\ldots, V_{-2}, V_{-1}, V_0, V_1, V_2, \ldots$ such that any node $v_i \in V_\ell \;\Leftrightarrow\; \lfloor \log_2 w_i \rfloor = \ell$. For each non-empty node-set we use

```
Delete-Table(T, s, v)
begin
  N ← size(T)
  S ← Σ_{i=0}^{N-1} |T[i]| − 1
  T[⌊s/N⌋] ← T[⌊s/N⌋] \ {v}
  if S = 0 then
    T ← empty
  else if S < N/4 then
    N ← N/2
    T'[0, ..., N_ℓ − 1] ← ∅
    for all v ∈ ⋃_{i=0}^{2N−1} T[i] do
      T'[⌊s/N⌋] ← T'[⌊s/N⌋] ∪ {v}
    od
    T ← T'
  fi
  return(T)
end.
```

Figure 5: Delete a node from a table

a table $T_i[0,..,N_i-1]$ of N_i elements where N_i is chose such that $N_i \leq |V_i| \leq 4N_i$. In this table we store at each entry $T_i[j]$ all nodes satisfying $s_v \in [j/N_i, (j+1)/N_i)$. These sets can be stored by a linked lists. Besides this, we provide a table T^* for all nodes organized as the other tables.

LEMMA 3.

1. *For $V_i \neq \emptyset$ a set $T_i[j]$ is empty with probability of at most $3/4$. The probability that an interval $T_i[j,\ldots,j+d]$ consists only of empty sets is at most $(\frac{3}{4})^d$.*

2. *The expected number of elements in the sets $T_i[j],\ldots,T_i[j+d]$ is at most $2d$ (even under the condition that the rightmost $d/2$ sets are empty).*

3. *If a data element x has d empty entries in T_ℓ left onwards from its position $p = \lfloor r_x \cdot N_\ell \rfloor$, i.e. $T_\ell[p-d+1],\ldots,T_\ell[p] = \emptyset$ and $T[p-d] \neq \emptyset$ then the corresponding node to x must lie in the in the sets $T_\ell[p-2d],\ldots,T_\ell[p-d]$ if it belongs to the layer ℓ.*

```
Lookup(x, T)
begin
  r_x ← h(x)
  H ← ∞
  ℓ_max ← max{i : V_i ≠ ∅}
  while ℓ ≥ ℓ_max − 2 log n − 1 do
    p ← ⌊r_x · N_ℓ⌋
    q ← p
    d ← 0
    while T_ℓ[q] = ∅ do
      q ← (q − 1) mod N_ℓ
      d ← d + 1
    od
    for i ← q − d − 1 to q do
      for all u ∈ T_ℓ[i mod N_ℓ] do
        s_v ← h(v)
        if H > D_{w_v}(r_x, s_v) then
          y ← v
          H ← D_{w_v}(r_x, s_v)
        fi
      od
    od
    ℓ ← ℓ − 1
  od
  p ← ⌊r_x · N*⌋
  d ← 0
  for all v ∈ T*[p] ∪ T*[p − q mod N*] do
    s_v ← h(u)
    if H > D_{w_v}(r_x, s_v) then
      y ← v
      H ← D_{w_v}(r_x, s_v)
    fi
  od
  return(y)
end.
```

Figure 6: Look up node y for data element x

Now let $d = H2^{-\ell}$ the expected number of nodes in this array is at most $2d$ and the chances to check so many entries is at most $(3/4)^d$. Summing over all d shows the expected running time to check one level T_i is constant. For the small weights $w_i \leq \frac{1}{n^2}\max_u\{w_u\}$ the probability to assign a data element to such a node is at most $\frac{1}{n^2}$. The probability that a data element is assigned to any such node is therefore at most $\frac{1}{n}$. Then, even linear time to detect this element leads to constant expected run time.

The amortized analysis of the Insert-Node and Delete-Node procedure uses standard techniques. □

By introducing q copies and k partitions, the needed space for this data structure rises to $\mathcal{O}(knq)$, while the running time is $\mathcal{O}(\log n + \log q)$ for data lookup and $\mathcal{O}(qk)$ for inserting or deleting a node.

4.5 The limits of the Linear Method

Consider $w_1 = 1$ and $w_2,\ldots,w_n = \frac{1}{n-1}$. The desired share for node 1 is $\frac{1}{2}$. It turns out that 1 receives more than this, if only one copy for the node positions is chosen.

THEOREM 5. *The Linear Method (without copies) for n nodes with weights $w_1 = 1$ and $w_2,\ldots,w_{n-1} = \frac{1}{n-1}$ assigns a data element with probability $1 - e^{-1} \approx 0.632$ to node 1 when n tends to infinity.*

PROOF. We use Lemma 1 and reduce the probability to the following term.

$$\lim_{n\to\infty} \int_{x=0}^{1} x\left(1 - \frac{x}{n-1}\right)^{n-1} dx =$$

$$\int_{x=0}^{1} xe^{-x}dx = \left[-e^{-x}\right]_0^1 = 1 - e^{-1}\ .$$

□

This shows that the Linear Method needs copies for improving the fairness. It is not enough to introduce partitions, although this leverages the size of intervals. So, the number of copies improve the quality of the Linear Method.

5. THE LOGARITHMIC METHOD

In this section we present the *Logarithmic Method*, it is usable for the same application areas. It uses similar techniques to assign data elements to storage devices, but different functions to concern their weights. So in the example, see Fig. 7*, one can see that the mapping range is again divided into several pieces, but slightly differs compared to Fig. 1. This yields in the example to diverse data assignments before and after the integration of the additional device v_5. Nevertheless the results in following subsections will show that this scheme also provides a fairly balanced distribution for data elements and is also capable to compensate dynamics with an minimal effort, concerning data movement or system reorganization.

*The vertical coordinate is compressed by applying the arctan with values from the hight function. This has no impact on the partition of the mapping range.

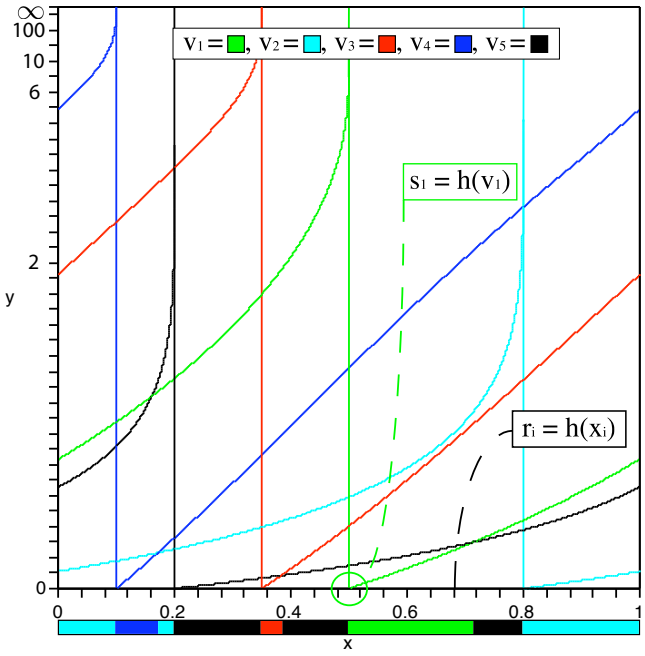

Figure 7: The Logarithmic Method

5.1 The Basic Method

Instead of a linear function we use $g(x) = -\ln(1-x)$ for the computation of the height. According to the *Logarithmic Method* a data element x is stored on the node v_i with weight w_i and hashed position $s_i = h(v_i)$ which minimizes the term $L_{w_i}(r_x, s_i)$ where

$$L_w(r,s) := \frac{-\ln((1-(r-s)) \bmod 1)}{w} ,$$

where $a \bmod 1 := a - \lfloor a \rfloor$. Again, for each node v and data element x we call the value of this function the height $H(z) := \min_v(-\ln((1-(z-s_v)) \bmod 1))/w_v$.

LEMMA 4. *Given n nodes with weights $w_1, \ldots, w_n$. Then the height $H(r)$ assigned to a position r in M is distributed as follows:*

$$P[H(r) > h] = e^{-\sum_{i \in V} w_i h} .$$

We observe for small h that this probability is close to the corresponding probability of the Linear Method. Furthermore, the probability $P[H^{\mathrm{Lin}}(r) > \ell]$ of the Linear Method tends also for larger h to this probability if we use many copies:

$$\lim_{\text{copies} \to \infty} P[H^{\mathrm{Lin}}(r) > \ell]^{1/\text{copies}} =$$

$$e^{-\sum_{i \in V} w_i h} = \mathrm{P}[H^{\mathrm{Log}}(r) > h] .$$

Since the Linear Method is fair when the number of copies grows towards an infinite number, this gives an alternative proof for the following theorem.

THEOREM 6. *Given n nodes with some positive weights $w_1, \ldots, w_n$ the Logarithmic Method assigns a data element to node v_i with probability $\frac{w_i}{\sum_{j=1}^{n} w_j}$.*

PROOF. Hence the probability that a data element receives height H_i in the interval $[h-\delta, h]$ and receives larger height than h is at most

$$\begin{aligned} \mathrm{P}[H_i \geq h-\delta \wedge H_i < h \wedge \bigwedge_{j \neq i} H_i \geq h] &= \\ \left(e^{-w_i(h-\delta)} - e^{-w_i h}\right) \prod_{j \neq i} e^{-w_j h} &= \\ e^{-w_i h}\left(e^{w_i \delta} - 1\right) \prod_{j \neq i} e^{-w_j h} &= \\ \left(e^{w_i \delta} - 1\right) \prod_{j \in [n]} e^{-w_j h} & \end{aligned}$$

The probability that an element is assigned to node v_i is upper-bounded by the following sum:

$$\begin{aligned} \lim_{\delta \to 0} \sum_{k=1}^{\infty} \left(e^{w_i k\delta} - 1\right) \prod_{j \in [n]} e^{-w_j \delta k} &\geq \\ \lim_{\delta \to 0} \sum_{k=1}^{\infty} k\delta w_i e^{-\sum_{j \in [n]} w_j \delta k} &= \\ \int_{x=0}^{\infty} x w_i e^{-\sum_{j \in [n]} w_j x} dx &= \frac{w_i}{\sum_{j \in [n]} w_j} \end{aligned}$$

The probability that a data element receives height in the interval $[h, h+\delta]$ and receives larger height than h is

$$\begin{aligned} \mathrm{P}[H_i \geq h \wedge H_i < h+\delta \wedge \bigwedge_{j \neq i} H_i \geq h] &= \\ \left(e^{-w_i(h)} - e^{-w_i(h+\delta)}\right) \prod_{j \neq i} e^{-w_j h} &= \\ e^{-w_i h}\left(1 - e^{-w_i \delta}\right) \prod_{j \neq i} e^{-w_j h} &= \\ \left(1 - e^{-w_i \delta}\right) \prod_{j \in [n]} e^{-w_j h} & \end{aligned}$$

The probability that an element is assigned to node v_i is lower-bounded by the following sum:

$$\begin{aligned} \lim_{\delta \to 0} \sum_{k=1}^{\infty} \left(1 - e^{-w_i k\delta}\right) \prod_{j \in [n]} e^{-w_j \delta k} &\leq \\ \lim_{\delta \to 0} \sum_{k=1}^{\infty} k\delta w_i e^{-\sum_{j \in [n]} w_j \delta k} &= \\ \int_{x=0}^{\infty} x w_i e^{-\sum_{j \in [n]} w_j x} dx &= \frac{w_i}{\sum_{j \in [n]} w_j} \end{aligned}$$

□

Similarly as in Theorem 1 this statement holds for n nodes and one data element inserted at the same time. In this situation, we achieve a perfect balance. If we insert more data elements, then we face strong dependencies between the assignments of the data elements. For the second element the probability that this element is inserted at some node v_i is highly dependent on whether the first element has been inserted at this node. This follows simply by the fact, that the intervals are fixed and their sizes determine the probability distributions.

5.2 Partitions

However, as in the Linear Method we can overcome this problem by using partitions and applying Chernoff bounds.

The nodes $V = \{v_1, \ldots, v_n\}$ are mapped to each of the continuous set $M_1, \ldots, M_k$ with $M_i = [(i-1)/k, i/k)$ and data elements $X = \{x_1, \ldots, x_m\}$ are mapped to one of the intervals in $[0,1)$ using hash functions. Then a data element is assigned to a node which is the closest in the sub-range $M_{\lfloor r_x/k \rfloor + 1}$ according to the logarithmic weighted height function.

THEOREM 7. *For all $\epsilon > 0$ and $c > 0$ there exists $c' > 0$, where we apply the Logarithmic Method with $c' \log n$ partitions. Then, the following holds with high probability, i.e. $1 - n^{-c}$.*

Every node $v_i \in V$ receives data elements with probability p_i such that

$$(1-\epsilon) \cdot \frac{w_i}{W} \;\leq\; p_i \;\leq\; (1+\epsilon) \cdot \frac{w_i}{W} \;.$$

PROOF. This result follows by Theorem 6 and applying Chernoff bounds. □

5.3 Data Structure

Here we can re-use the data structure of the Linear Method.

THEOREM 8. *There is an algorithm that determines for a data element the corresponding node according the Linear Method in expected time $\mathcal{O}(\log n)$. The data structure has size $\mathcal{O}(n)$. Inserting and deleting nodes in this data structures needs amortized time $\mathcal{O}(1)$.*

PROOF. We can re-use the data structure presented for the Linear Method. We just replace all occurrences of the height function D_v by $L_w(r,s) := \frac{-\ln((1-(r-s)) \bmod 1)}{w}$. One can show that Lemma 3 also holds for the Logarithmic Method as well as for the low weighted nodes, the same run time analysis is valid. □

6. FURTHER TECHNIQUES

6.1 Fragmentation

For some applications like Storage Area Networks there is only one hash function for the nodes, i.e. storage devices. The data elements are continuously placed in the interval. Then, it is interesting to count the number of intervals of data elements assigned to a node, called fragments. The fragmentation counts this number of intervals.

THEOREM 9. *The Linear Method with q copies and k partitions has a fragmentation of qkn for n nodes. The Logarithmic Method using k partitions has a fragmentation of $2kn - 1$.*

PROOF. This theorem follows by applying results from Davenport Schinzel Sequences [1]. □

The standard choice of parameters is $q = \mathcal{O}(1)$ and $k = \mathcal{O}(\log n)$ to achieve constant precision with high probability. Therefore both methods provide a fragmentation of $\mathcal{O}(n \log n)$.

6.2 Double Hash Functions

If fragmentation is not an issue and if the number of nodes is small, then *double hash functions* are an interesting extension of the Linear and Logarithmic Method.

For this, we apply for each node an individual hash function $h : V \times [0,1) \to [0,1)$. So, we start mapping the data element x to $r_x \in [0,1)$ as above and then for every node we compute $r_{i,x} = h(i, r_x)$. Now x is assigned to a node v_i which minimizes $r_{i,x}/w_i$ according the Linear Method. In the Logarithmic Method x is assigned to the node minimizing $-\ln(1 - r_{i,x})/w_i$.

The main advantage of this method is that it achieves the same probability distribution as the Linear Method, resp. Logarithmic Method with a large number of partitions. The drawback is (intrinsic) linear running time $\mathcal{O}(n)$ to determine the node a data element is assigned to.

THEOREM 10. *The Linear Method using double hash functions assigns data elements to all of the n nodes with probabilities p_i for each node, such that*

$$(1-\sqrt{\epsilon}) \cdot \frac{w_i}{W} \;\leq\; p_i \;\leq\; (1+\epsilon) \cdot \frac{w_i}{W} \;.$$

The Logarithmic Method using double hash functions assigns data elements to all of the n nodes with probabilities p_i for each node, such that

$$p_i = \frac{w_i}{W} \;.$$

PROOF. follows by the proof of Theorem 2 and Theorem 6 using the fact that now the heights of the data elements are independently distributed. □

6.3 Predicting Migration

We have defined the height H_x of a data element x for finding its corresponding node. There is a further property we like to point out. This height is proportional to the probability that this data element is moved, if a new node arrives.

FACT 1. *If in the Linear Method (without copies and without partitions) a node arrives with weight w, then the probability that data element x with previous height H_x is assigned to the new node is* $\min\{H_x w, 1\}$.

This fact follows by calculating the length of the interval describing possible positions where the new node receives this data element. In the Logarithmic Method the situation is similar:

FACT 2. *If in the Logarithmic Method (without copies and without partitions) a node arrives with weight w then the probability that data element x with previous height H_x is assigned to the new node is $1 - e^{-wH_x}$.*

Note that for small values $H_x \ll \frac{1}{w}$ the term $1 - e^{-wH_x}$ can be approximated by wH_x.

So, the probability that a data element needs to be assigned to a new arriving node can be calculated in advance. Note that the order of these reassignment probabilities is independent from the weight of the arriving node. This feature can be used for predicting data migration and optimizing data storage for this purpose.

6.4 Fading and Adaption

An interesting feature of the proposed weighted consistent hashing is the chance of smoothly fading in of new nodes without any overhead (according to data re-assignments). This avoids allocating a large chunk of storage for a new

node in one step. The alternative is a new node starts with a very small weight $w = \epsilon$ and slowly increases this weight. If this is the only node entering the system then only data will be assigned to this node which anyway would end there. So, bursty traffic can be avoided this way and the transmission bandwidth can be used more efficiently. Another advantage is that during the insertion of the new node the system state is always defined and in every time step only a little portion of data needs to be kept on two nodes.

Of course, the inverse function of fading out can be used, too. If a node is taken out of the system, then depending on the available bandwidth this node slowly reduces its weight towards zero weight.

Along this line nodes can use the weight to adapt the storage usage. However, such distributed algorithms need to be designed carefully since all nodes receives some relative portion $w_i/\sum_j w_j$. E.g. if all nodes increase their weight synchronously then nothing changes*.

6.5 Other height functions and range spaces

We have proposed the linear and the logarithmic height function in one-dimensional ring to achieve a weighted version of distributed hash tables. One might asks what happens if we use different height measures or two-, three- or higher-dimensional space.

Therefore we shortly discuss the following modifications. It is a straight-forward observation that a polynomial height function for some $\beta > 0$:

$$D_w(r,s) := \left(\frac{r-s}{w} \bmod 1\right)^\beta$$

as a replacement for the linear height function does not change anything. This follows from the fact that for determining the minimum height we apply the minimum function and the outcome remains the same. Of course any other monotonic growing function applied to $\frac{r-s}{w} \bmod 1$ leads to the same situation.

On first sight the situation seems to be more interesting if we map data elements and nodes to a two-dimensional space $[0,1]^2$ and use a distance function. However one can reduce the probability distribution of a node receiving a certain height to a quadratic height function in one-dimensional space. So, for the balance we end up with the Linear Method in the one-dimensional space $[0,1)$. Analogously, it turns out that for every constant dimensional space and distance measure according to the probability distribution there the situation is more or less the same as in the Linear Method.

7. CONCLUSIONS

We present two elegant, simple methods for weighted consistent hashing that go along the original method of [9], called the Linear Method and the Logarithmic Method. They are efficient, easy to implement, and applicable to many fields where up to now only uniform distributed hash tables are used. Both methods overcome problems in approaches presented so far for weighted distributed hash tables.

The plain Linear Method assigns a data element to a node v_i with probability of at most $w_i/\sum_{j\neq i} w_j$. If used with $q = \lceil 2/\epsilon \rceil + 1$ copies of all nodes, one can show that this probability is at least a factor of $1-\sqrt{\epsilon}$ larger and at most a factor of $1+\epsilon$ smaller than the desired probability $w_i/\sum_j w_j$. To apply this method for many data elements with high probability, we partition the range space into $\mathcal{O}(\log n)$ sub-spaces, called partitions. We can show that then with high probability all data elements are assigned to all nodes according to this probability distribution. This increases fragmentation up to $\mathcal{O}(n/\epsilon^2 \log n)$, and provides an $1 \pm \epsilon$ approximation of the probability $w_i/\sum_j w_j$. We are using this method in a Peer-to-Peer network storing routing information for a mobile ad hoc network in [2]. There is an efficient data structure that provides for each data element the corresponding peer in time $\mathcal{O}(\log n)$. Furthermore, we present an efficient data structure of size $\mathcal{O}(n)$ for n nodes which allows to compute the assignment for each data element in expected time $\mathcal{O}(\log n)$. Insertion and deletion of nodes in this data structure can be handled in amortized time $\mathcal{O}(1)$.

The Logarithmic Method is fairer than the Linear Method. The plain Logarithmic Method assigns a data element to node v_i with probability $w_i/\sum_j w_j$. Using $O(\log n)$ partitions this scheme can handle more data elements. With high probability it assigns them with a constant factor approximation of the balanced probability distribution. It turns out that the data structure designed for the Linear Method can also be used for this Logarithmic Method with the same time and space resources.

*Note that in this case also no data elements are reassigned.

8. REFERENCES

[1] P. K. Agarwal and M. Sharir. Simple bounds. In *Davenport-Schinzel Sequences and Their Geometric Applications*, chapter 1, pages 1–47. Cambridge University Press, in handbook of computational geometry edition, 1995. ISBN 0-521-47025-0.

[2] P. Bleckmann, S. Böttcher, E. Cesnavicius, A. Francisco, T. D. Hollerung, B. Kühnel, M. J. Liu, S. Obermeier, S. Oberthür, F. Peter, F. Rammig, C. Schindelhauer, G. Schomaker, T. Steenweg, Q. A. Tarar, M. Tiemeyer, A. Türling, and A. Vater. The design of pamanet the paderborn mobile ad-hoc network. In *MobiWac '04: Proceedings of the second international workshop on Mobility management & wireless access protocols*, pages 119–121, New York, NY, USA, 2004. ACM Press.

[3] A. Brinkmann, F. Meyer auf der Heide, K. Salzwedel, C. Scheideler, M. Vodisek, and U. Rückert. Storage management as means to cope with exponential information growth. In *Proceedings of SSGRR 2003*, L'Aquila, Italy, 28 July - 3 Aug. 2003.

[4] A. Brinkmann, K. Salzwedel, and C. Scheideler. Efficient, distributed data placement strategies for storage area networks (extended abstract). In *Proceedings of the twelfth annual ACM symposium on Parallel algorithms and architectures*, pages 119–128. ACM Press, 2000.

[5] 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)*, pages 53–62, Winnipeg, Manitoba, Canada, 11 - 13 Aug. 2002.

[6] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. Technical report, BU Computer Science, 2002.

[7] P. Druschel and A. Rowstron. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In R. Guerraoui, editor, *Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2001.

[8] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of the 14th Annual ACM Symposium on Parallel ALgorithms and Architectures (SPAA-02)*, pages 41–52, New York, Aug. 10–13 2002. ACM Press.

[9] 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 *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 654–663, El Paso, Texas, 4–6 May 1997.

[10] D. R. Karger and M. F. Kaashoek. Koorde: A simple degree-optimal distributed hash table. *In Proc. 2nd IPTPS, Berkeley, CA, Feb. 2003*, Feb. 10 2003.

[11] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

[12] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 50–59. ACM Press, 2003.

[13] C. G. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '97)*, pages 311–320, New York, June 1997. Association for Computing Machinery.

[14] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. *In 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Computer Communication Review*, volume 31, pages 161–172. Dept. of Elec. Eng. and Comp. Sci., University of California, Berkeley, 2001.

[16] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In R. Guerin, editor, *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, volume 31, 4 of *Computer Communication Review*, pages 149–160, New York, Aug. 27–31 2001. ACM Press.

[17] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, Apr. 2001.