First publ. in: IEEE Computer society, 32nd IEEE Conference on Local Computer Networks, 2007, pp.315-324
http://doi.ieeecomputersociety.org/10.1109/LCN.2007.47

Replica Placement and Location using Distributed Hash Tables

Daniel Bauer Paul Hurley
IBM Research GmbH, Zurich Research Laboratory
Säumerstrasse 4 / Postfach
8803 Rüschlikon, Switzerland
{dnb,pah}@zurich.ibm.com

Marcel Waldvogel
University of Konstanz
78457 Konstanz, Germany
Marcel.Waldvogel@uni-konstanz.de

***Abstract*—Interest in distributed storage is fueled by demand for reliability and resilience combined with decreasing hardware costs. Peer-to-peer storage networks based on distributed hash tables are attractive for their efficient use of resources and resulting performance. The placement and subsequent efficient location of replicas in such systems remain open problems, especially (1) the requirement to update replicated content, (2) working in the absence of global information, and (3) determination of the locations in a dynamic system without introducing single points of failure.**

We present and evaluate a novel and versatile technique, *replica enumeration*, which allows for controlled replication and replica access. The possibility of enumerating and addressing individual replicas allows dynamic updates as well as superior performance without burdening the network with state information, yet taking advantage of locality information when available. We simulate, analyze, and prove properties of the system, and discuss some applications.

I. Introduction

Peer-to-peer (P2P) systems offer enormous potential. While some may still equate it with music piracy, the technology itself and its many uses are entirely content-neutral. In fact, some IT specialists hope that peer services will provide a method for matching the increasing reliance of business and government processes on continuous, instantaneous, and reliable access to data. It has also been argued, that heterogeneous, physically distributed systems are more resilient to attacks using physical force against servers as well as intrusion and distributed denial-of-service (DDoS) attacks. Recent advancements in peer technologies, such as the introduction of the Distributed Hash Table (DHT) concept, have caused a noticeable shift from unreliable toy systems to scalable enterprise-level services capable of managing global data repositories. In contrast to the previously used technique of extensive flooding of the network with queries, DHT enable the efficient addressing of data at unknown locations using only the unique *resource (document) ID*. A distributed mapping from a resource ID to a set of hosts with minimal routing information in each node is produced.

The availability of multiple replicas is crucial in order to provide resilience and support high demands for certain resources. Throughout, we assume each resource exists at least as an *original*, with potentially further *copies* (*mirrored*, *replicated*, or *cached*). The original and its copies will be jointly referred to as *instances* of a particular resource, all kept by *holders*.

In this paper, we present and evaluate, in detail, a novel technique, called *replica enumeration* (RE), which allows end-systems to make an informed decision about where replicas are and which ones to access. It enables updates to reach all replicas easily, thereby allowing updates to be pushed immediately, without having to wait for the next synchronization initiated by the replica holder. RE is achieved without explicit meta-data or the need for a replica directory service,[1] works as an independent layer on top of any DHT, and suits the "dumb network" and "local knowledge only" paradigms associated with the Internet and P2P networks.

For each document, RE places replicas at the DHT addresses determined by a two-variable hash of the document id and the index of its replica instance. As we shall show, facilitating the location of the closest replica is achieved through simple rules regulating replica placement.

RE is well suited to a globally distributed storage system, but also for applications in locally confined environments. Examples include its use for load balancing in a server farm or as a backup for the company-wide centralized replica directory. Other applications include distributed directory services or databases, where location speed becomes critical and retrieved data relatively small.

We analyze and prove the properties of RE and provide simulation results. As part of our evaluation, we also present classification criteria for replica management and discuss the trade-offs of different mechanisms.

A. Paper Organization

The paper is organized as follows. Section II introduces background and related work. Section III presents RE in detail. Section IV shows the results of our simulations. Section V describes applications and combinations of RE. Section VI concludes the paper, and presents further work.

II. Background and Related Work

A. Distributed Hash Tables

Several scalable overlay networks have recently sprung to life. Sitting "on top" of the Internet, they add additional value to the transport capabilities of the underlying network. DHT provides a mapping from resource IDs to a set of hosts ($d \rightarrow$

[1]…which in itself would need to be replicated and thus raise the demand for a meta-replica directory service, ad infinitum.

$\mathcal{H}$) that is typically preceded by a mapping from resource name to resource ID ($N \rightarrow d$). DHTs are generally designed to use minimal routing information in each node and to deal with changes in host availability and network connectivity.

A toy example of a DHT using rectangular subdivisions of a two-dimensional ID coordinate space is illustrated in Figure 1, each rectangle with its manager, represented by a filled dot. When hosts join, they are assigned an ID in the same coordinate space as the resources. The newcomer arranges a split of the rectangle it is located in with the incumbent manager, assuring that each rectangle always has exactly one manager. To forward messages in such a system, each node only needs to know its direct neighbors. Assume the bottom left host would like to retrieve the resource whose ID corresponds to the hollow circle, then it sends (as outlined by the arrows) the request to the neighboring manager whose rectangle is closest to the destination, which in turn repeats the process, until the host responsible for the resource has been reached. This process ensures that every resource is reachable efficiently, even when hosts hold only limited routing information.[2]

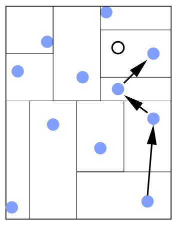
Fig. 1. DHT based on 2-D rectangular subdivision

Real DHTs come in a variety of routing flavors, but all share the property that messages are transported on a hop-by-hop basis among constituent nodes of the overlay network. Each hop passes the message closer to the destination, until it finally reaches the node that claims the requested ID.

Some DHTs operate based on intervals in skip-list-based ring topologies (Chord [1], [2], SkipNet [3]), some split hyperspaces into manageable chunks (CAN [4], Mithos [5]), whereas others are probably best described as a rootless tree implementation (P-GRID [6], Pastry [7], Tapestry [8])

Many of these DHT systems are able to exploit the locality of the underlying network. Locality aspects are typically separated into *geographic layout* and *proximity forwarding*, categories adapted from [9]. Moreover, a node that knows more than another node will bring the message closer to its ultimate destination, generally calculated in the resource ID space. From among these nodes, proximity forwarding selects one that is close by also in terms of the underlying network (this is implemented in Chord and Pastry, among others).

Geographic layout, on the other hand, implies that the ID space is already partitioned into regions ("hash buckets") based on the connectivity in the underlay, a method utilized by CAN and Mithos. Geographical layout automatically implies some form of proximity forwarding.

B. Replication

Traditionally, mirroring has been used for replication. Mirrors typically know about all or a subset of the other mirrors, in order to be able to point at some of them. While this redirection can be automated , it is most often performed manually. Criteria for selecting a mirror may include reliability, access latency, or throughput. To maintain data consistency, there often exists a relationship between the master and the mirrors. Elaborate mirroring-based systems include Usenet News, Akamai [10], Lotus Notes and IMAP.

A less organized method for replication includes caching, which is widely used and researched in particular for web caches (e.g. [11]). Here, consistency is achieved by a combination of the server stamping expiration dates and regular update queries to the server. The caches are sometimes organized into semi-hierarchical topologies to improve performance and reduce server load.

Content distribution networks (CDNs) [12] are sets of interoperating caches that replicate documents to places where user demand is high. Requests are mapped to close-by caches using DNS servers that take server- and network load into account. Consistency is maintained using time-stamping, version numbering, and on-demand purges (invalidations) of content. The ability to assemble documents dynamically allows CDNs to cache otherwise uncacheable content.

One of the most organized ways of linking DHTs and caching is employed by OceanStore [13]. When there is a high probability that a cached copy of the document can be found along that route, DHT queries are redirected by Attenuated Bloom Filters (ABF). In addition to the possibility of false positives despite continuous ABF update traffic, there is no way for the document originator to address selected (or update all) replicas when the need arises.

Beehive [14] is a replication framework that provides constant lookup times for query distributions that follow a power law distribution. It works on top of DHTs that are based on prefix routing such as Chord, Pastry or Tapestry. Using an analytical model, Beehive divides the set of objects into groups based on their popularity. Objects with high popularity are replicated such that they can be found using a shorter prefix; less popular objects need a longer prefix. Beehive minimizes the number of replicas for a specified average prefix length. Since prefix length is directly proportional to the number of hops required to access an object, Beehive is able to provide access in a constant number of hops.

The distributed hash (DHash) component of the Cooperative File System [15] offers both caching and replication. It is based on Chord. Documents, stored on a "home" server, are replicated to a number of successors on the Chord ring. The home server maintains meta-information, including the identifies of the servers that hold replicas as well as latency information. Clients retrieve this information from the home server and choose the replica that is accessible with the lowest latency. In addition, DHash caches documents along the lookup path, using a least-recently-used strategy. DHash's cache mechanism may result in stale documents since updates only affect replicated, but not cached documents.

Specialised cases include (1) Dynamic Replication [16], which is based on our original idea [17] but requires knowledge of the underlying structure and fine-tuning to work while (2) Harvesf and Blough [18] try to create disjoint routes to replicas.

Another approach is followed by Global Internet Anycast

[2]For easy of understanding, leave operations and outage handling are omitted.

TABLE I
COMPARISON ACCORDING TO CRITERIA CATALOG OF SECTION II-C

[3] Criteria	Fixed	OceanSt.	Proxy	Beehive	CDN	GIA	RE
Openness	all	near path	locals	yes	all	nearby	all
Locality	manual	yes	yes[†]	yes[‡]	yes	yes[*]	often
Addressability	yes	no	no	yes	no	no	yes
Freshness	yes	no	no	yes	yes	N/A	yes
Adaptivity	no	yes	yes	no	yes	yes	yes
Flexibility	no	yes	yes	yes	yes	yes	yes
Variability	no	yes	no	no	no	yes	no
State size	full	ABF	minimal	minimal	full	medium[*]	minimal
Resilience	yes	yes	yes	yes	yes	partial	yes
Independence	no	no	yes	no	yes	yes	yes
Performance	no impact	redirect	no impact	no impact	no impact	measures	local

The properties of caching proxies also apply to local replicas used in systems such as Lotus Notes or IMAP. Recall that adaptivity and flexibility apply to the replica count whereas variability refers to the location.
[†] Only for recently-requested (cached) documents.
[‡] Hop-count in underlying DHT is minimized.
[*] Only for frequently-requested documents.

(GIA) [19], which uses the BGP routing infrastructure and in-router traffic analysis to identify the frequently used documents. Read-only accesses will then dynamically be redirected to a nearby copy instead of the original, if the access rate exceeds a given threshold. Updates are not supported by the system and require the use of an out-of-band mechanism.

Replication in general *unstructured* networks has been analyzed by Cohen and Shenker [20]. They show that the inherent use of flooding in these networks discourages frequent replication. Their result is not applicable to the directed queries used in DHTs. Additionally, analysis of resource replication in the event of *selfish server nodes* is given in [21]. APRE[22] uses probabilistic search to achieve dynamic replication in unstructured networks.

C. Replication Criteria

In order to help evaluate different forms of maintaining multiple instances, we will use the following list of criteria for the evaluation of replica management and access.

Openness. Replicas should be useful to many requesters, not only a single user.

Locality. Obtaining a "nearby" replica is preferable. The actual distance (or cost) metric used may include dynamic parameters such as network and server load.

Addressability. For management, control, and updates, support should be provided for enumeration and individual or group-wise addressing of replicas.

Freshness. Each replica access should yield the freshest document version.

Adaptivity. The number of replicas for a resource should be adaptable to demand, as a trade-off between storage requirement and server load.

Flexibility. The number of replicas for one resource should not depend on the number of replicas for another.

Variability. The locations of replicas should be selectable.

State size. The additional state, both distributed and centralized, required to maintain the replicas should be minimum.

Resilience. As DHTs themselves are completely distributed and resilient to outages, centralized state or other single points of failure should be avoided.

Independence. The introduction of a new replica (respectively, the removal of an existing replica) on a node should depend on as few other nodes as possible.

Performance. Locating a replica should not cause excessive traffic or delays.

Our models assume that, in this scenario, communication is far more expensive than local computation.

D. Comparison

Table I compares RE with other replication policies, such as local replicas or local caching proxies, the ABF used by OceanStore, and replication to fixed, hard-coded locations, as is often used in a database context.

Fixed installations often come with hard-coded replica references, resulting in a system without communication overhead that uses locality if correctly configured, but is highly resistant to configuration changes, requiring manual intervention. Adding a directory service leads to a trade-off between performance impact and update frequency; typically the locality property is also lost in the process.

OceanStore is able to place copies wherever needed, but only queries that pass close to a cached copy will use it. The downsides include (1) potentially significant traffic volume due to frequent exchanges of bloom filters, (2) the possibility of increased forwarding cost to the destination resulting from misleading redirection caused by the ABF, and (3) the impossibility of updating or ensuring copy freshness.

Beehive is able to provide lookups with a constant average path length, the drawback is that hop count in a DHT is often unrelated to the network latency. In addition, Beehive is restricted to prefix-based DHTs only; the location of where replicas are stored is inherently coupled with prefix-based routing.

DHash is tightly integrated with Chord and does not work on top of other DHTs. Unless a cache-hit occurs, clients first access the home server in order to get the list of replicas, which limits the benefits of accessing the replica with the lowest latency. The location of replicas is not selectable. Furthermore, while replicas are updated when content changes, cached documents are not and access to stale documents is possible.

Caching proxies and local replicas are only accessible by local users (often just a single user) and therefore cannot be shared unless combined with elaborate inter-cache communication schemes, which introduce significant overhead. They cannot be kept up to date or be notified of updates.

CDNs optimize access to read-only documents. The overhead in these systems is rather high; a monitoring system constantly measures server- and network load and feeds the results into a DNS-server based control infrastructure that keeps state information about replicas. State information in the DNS servers is constantly updated in order to avoid inconsistencies. Document replicas are not directly addressable, the matching between client and replica is instead done by a third party, the DNS server. Changes are allowed to propagate from the original server only.

GIA is similar to CDNs, but uses the routing infrastructure instead of DNS.

One weakness of *RE* is that it evenly distributes replicas among the network nodes, thus not satisfying the variability criteria. However, given the current level of network backbone connectivity and capacity, even a small number of replicas will lead to a configuration where at least one replica is sufficiently close. Additionally, we believe that enhancements in RE that tailor it to an estimate of the current node distribution are likely to further improve the probability of finding a replica nearby. This remains an item for future work.

For applications that require very low latency, combinations with other schemes, such as those in Section V, may prove helpful. The high conformance of RE to the other criteria is another strong point in its favor. Note that several of RE's advantageous properties are provided or at least facilitated by the underlying DHT. However, they are still advantages of RE, as DHTs without RE lack these properties.

III. Replica Enumeration

Document replica placement in large-scale environments has traditionally been driven by demand, either from a single user or a small group. Caching proxies, for example, are placed based on decisions of the local user community, independently of where other caching proxies are. RE, on the other hand, uses a coordinated approach based on a globally known algorithm for placement.

The basic idea behind RE is simple: For each document with ID d, the replicas are placed at the DHT addresses determined by $h(m, d)$, where $m \geq 1$ is the index, or number, of that particular instance, and $h(\cdot, \cdot)$ is the allocation function, typically based on a hash function shared by all nodes.[4] The allocation function may, for example, be defined to include the original resource at its existing location d in the search as follows: $h(1, d) = d$; $h(m, d) = H(m||d)$, where $H(\cdot)$ is a hash function and $||$ is concatenation.

Listing 1 ADDITION

```
/* Triggered by high load */
Determine $r_d$ and atomically lock $h(r_d, d)$;
Create replica at $h(r_d + 1, d)$, ignore existing-replica errors;
Release lock on $h(r_d, d)$;
```

Listing 2 DELETION

```
/* Run at replica having an underutilized document d */
Determine the instance index, $m$, for this replicated document;
Exclusively lock the document $h(m, d)$;
if exists $h(m + 1, d)$ /* Are we the last replica? */ then
    /* Cannot remove replica, would break rule 3 */
else
    Remove local replica;
end if
Release lock on $h(m, d)$;
```

Thus, any node in the network can easily determine any potential replica address. However, knowing only the potential addresses is not sufficient. To access a replica, the number of replicas actually present or the location of the closest replica is needed.

To this end, following four simple replica-placement rules govern basic system behavior:

1) Replicas are placed only at addresses given by $h(m, d)$.
2) For any document d in the system, there always exists an initial replica with $m = 1$ at $h(1, d)$.
3) Any further replica ($m > 1$) can only exist if a replica currently exists for $m - 1$.
4) No resource exists in more than R instances.

The first three of the above invariant rules indicate that for a document d with r_d replicas, the replicas will be placed at $h(m, d)$, where $m \in [1, r_d]$, resulting in a contiguous set of values for m. The only system parameters that need to be pre-agreed upon and remain static are the choice of hash function $h(\cdot, \cdot)$ and the maximum number of replicas, R. With these rules in place, the actual number of replicas currently present is not needed to perform the most common operation, namely lookup. Before introducing and tuning the lookup algorithm, replica addition and deletion are, respectively, presented in Listings 1 and 2.

Locking is used to simplify the algorithm and render it deterministic. Verification of rule 3 after a lock-free addition/ deletion can be used to recover from temporary inconsistencies. RE does not require permanent consistency. In fact, there may be temporary inconsistencies if r_d is modified in the middle of a lookup, resulting, possibly, in some topmost replica(s) being not considered for retrieval.

[4]We assume that $h(\cdot, \cdot)$ is pseudo-random, uniformly distributing the DHT addresses of the replicas. This is not required for system correctness but performance would need to be re-evaluated.

Listing 3 AWARE: Location-aware replica selection

```
/* Locate a replica for document ID d */
$r \leftarrow R$;
/* Calculate cost for each potential replica */
$\forall i \in [1, R] : c_i \leftarrow cost(h(i, d))$;
while $r \geq 1$ do
    $m \leftarrow$ index of minimal cost among $c_i$, $(i \leq r)$;
    Request document with ID $h(m, d)$;
    if request was successful then
        return document;
    end if
    $r \leftarrow m - 1$;
end while
return nil;
```

The actual mechanisms used to decide on addition/deletion of a replica are orthogonal to, and outside the scope of, this paper. Examples include any of the replica holders being overloaded, the topmost replica holder being short on storage, or a query node finding performance inadequate. Each of these nodes can then initiate the appropriate measures.

A. Basic Lookup: Locality-Aware, Reliable Case

Several DHT systems, such as CAN, Mithos and SkipNet, support some notion of locality, i.e., from the DHT address, any device can estimate if the destination address is likely to be close. Also, there are several other systems that allow distance determination that can be used in conjunction with DHTs (e.g. [23], [24]).

Assumptions: For a first description, let us assume the existence of a reliable DHT with support for locality; i.e. for two addresses returned by the hash function $h(\cdot, \cdot)$, any device is able to decided which of the two is closer. We will later generalize the system to include unreliable behavior, lack of *a priori* knowledge of distances, and faster convergence. We also assume initially that distance information is perfect. Local computations are considered to be much cheaper than communications, and, for ease of analysis, the assumption that the hash function $h(\cdot, \cdot)$ conforms to an essentially uniformly random distribution.

The basic algorithm is similar in spirit to a binary search on the initial range $[1, R]$, with a different pivot element selection process. Instead of picking the middle element, as in ordinary binary search, the node m with the least cost is picked and probed. If it contains the replica, no further probing is necessary. Otherwise, rule 3 that $r_d < m$ means that the search range can be reduced to $[1, m-1]$. The process is then repeated with the resulting narrower span until it succeeds. Success is guaranteed because of rule 2, unless, of course, the document has not been stored in the DHT at all. The search is also explained in algorithmic form in Listing 3. The expected number of probing rounds required is approximately logarithmic with the maximum number of replicas R. This is proved in Section III-B.

Figure 2 provides an example for $R = 12$. The potential replicas are shown as vertical bars, with increasing instance indices from left to right. The length of the bar indicates the distance to the querying node. The first five nodes (in the dashed rectangle) contain a replica. The closest three nodes are labeled A to C, in order of increasing distance. The algorithm is then:

1) The closest node, A, is probed, but fails.
2) The range shrinks to only the first six nodes, up to but excluding A. The second closest, B, thereby is also excluded.
3) The closest node remaining in the reduced range, C, is probed and succeeds.

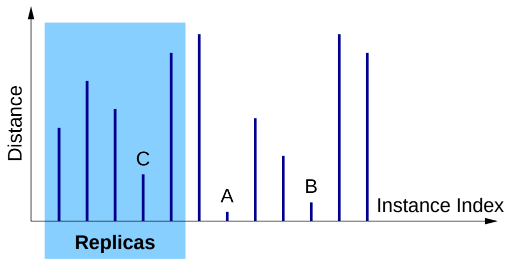

Fig. 2. Lookup Probing Strategy ($R = 12$, $r_d = 5$)

If large proportions of the population were to attempt to bypass the protocol and directly access the replica with index 1, that node could be quickly overloaded.As they would mostly harm other "cheaters," there seems to be little incentive to take the shortcut. Also, in a location-aware system, replica number 1 might be more costly to access than the optimal replica, even including the search phase.

B. System Properties

This section will provide correctness proofs and performance analysis and show that AWARE will find the closest instance if at least one exists. Closed form solutions for the probability distribution, expected value, and variance of the number of rounds necessary to find a replica are derived (equations (1), (4) and (5) respectively), as well as approximations for the expected value and variance.

Before diving into the details, we would like to provide an understanding of the number of rounds required as a consequence of the probability distribution in (1). Figure 3 considers the example where there are a maximum of $R = 100$ replicas and shows the distributions when the number of replicas of the document is $1, 2$ or 10, when no replicas of the document being searched for are added or deleted during a search operation. We see that the expected number of probing rounds is very small relative to the maximum number of replicas – roughly $\log(R/r_d)$.

Although some pathological cases do exist (such as linear traversal of all possible replicas when the cost function degrades into a decreasing function of the instance index), they have almost zero probability of occurring. For example, when there is only one replica of the document ($r_d = 1$), there is a probability of 99.9% of taking no more than 13 rounds to obtain the document.[5]

[5]By adding a delayed, logarithmically sweeping upper bound, the worst case can be reliably bounded at the cost of not considering a few replicas in the worst case.

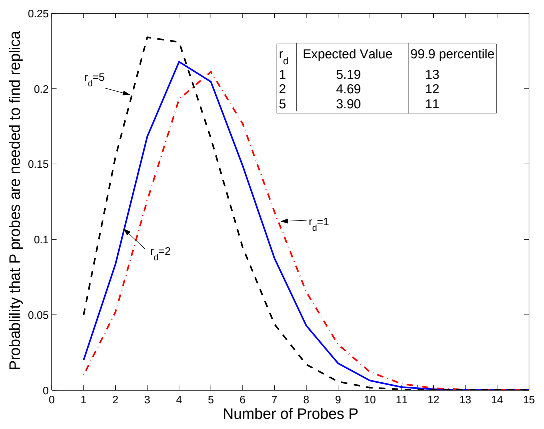

Fig. 3. Probability distributions of the number of rounds required in order to find the document, for cases where there are $r_d = 1, 2, 5$ replicas of it. The expected number of rounds needed is also shown as well as the 99.9th percentile (e.g. when $r_d = 1$, 99.9% of all document searches will find the document with no more than 13 rounds).

The following descriptions all assume proper operation of the underlying DHT, such as its ability to route all messages and its ensuring of document continuity under reconfiguration. Many DHTs include support for providing reliability. Still, Section III-F describes RE operation in the presence of faults.

Lemma 3.1: Assume that replicas of a document neither arrive nor leave the system during a search, that a scalar cost function *cost*$(h(\cdot,\cdot))$ is defined, and at least one replica exists. Then AWARE will find *the closest* one, where closest is defined according to the cost function.

Proof: The algorithm closely follows binary search: Start with an initial range covering all possible matches and only remove those parts that are guaranteed not to contain a match. In a reliable system, a probe at instance index m either succeeds, returning the document, or fails because of its absence. If a probe failed, rule 3 signifies that no replica $\geq m$ can exist, and thus the upper end of the range can be lowered to $m-1$ without risk of missing an entry. As the range ("potential instances") at any time includes at least all actual instances, the closest of all *potential* instances must be at least as good as the closest *actual* instance. By choosing the closest in-range (potential) instance for the next probe, it is impossible to miss the closest actual instance. ■

Obviously, a cost function providing only estimates is still valuable, but the quality of the replica returned will be bounded by the quality of the cost function.

C. Analysis

The analysis gathers insight into the algorithm through a complete probability distribution for the number of probes and exact average case and variance results through the use of generating functions. Alternative powerful techniques, such as the worst-case analysis methods in [25], [26] may also be of interest. The following lemma enables us to establish the probability distribution of the number of probes needed by the location algorithm.

Lemma 3.2: Let R be the maximum number of replicas, and let the document being searched for be replicated $r_d \geq 1$ times. Assume no replicas of it arrive or leave the system during the search. Let S be the round number where the document is found. Upon application of the random searching algorithm, the probability of finding a document after s probes is given by:

$$P(S=s) = \frac{r_d}{R}\sum_{x_1=r_d}^{R-1}\sum_{x_2=r_d}^{x_1-1}\cdots\sum_{x_{s-1}=r_d}^{x_{s-2}-1}\frac{1}{x_1x_2\ldots x_{s-2}x_{s-1}} \qquad (1)$$

where $s = 1, 2, 3, \ldots, R - r_d + 1$.

Proof: At the ith probe, let the replica number chosen to be probed be the random variable X_i. A successful document search happens when $X_i \leq r_d$.

If s is the first probe such that $X_s \leq r_d$, then necessarily $X_i > r_d$ for all $i < s$ and,

$$\begin{aligned}P(S=s) =& P(X_s \leq r_d \wedge \forall_{i=1}^{s-1} X_i > r_d)\\ =& P(X_1 > r_d)P(X_2 > r_d|X_1 > r_d)\ldots\\ & P(X_s \leq r_d|X_1 > r_d \ldots X_{s-1} > r_d).\end{aligned}$$

Now, the event $\{X_1 > r_d\}$ occurs if and only if $X_1 = x_1$ for some $x_1 \in \{r_d + 1, \ldots, R\}$. Similarly, for any $i = 2, 3, \ldots$ and conditional on $X_1 = x_1, X_2 = x_2, \ldots X_{i-1} = x_{i-1}$, the event $\{X_i > r_d\}$ occurs if and only if $X_i = x_i$ for some $x_i \in \{r_d + 1, \ldots, x_{i-1} - 1\}$. Hence,

$$\begin{aligned}P(S=s) = \sum_{x_1=r_d+1}^{R}\frac{1}{R}\sum_{x_2=r_d+1}^{x_1}\frac{1}{x_1-1}\sum_{x_3=r_d+1}^{x_2}\frac{1}{x_2-1}\cdots\\ \sum_{x_{s-1}=r_d+1}^{x_{s-2}}\frac{r_d}{x_{s-2}-1}\sum_{x_s=1}^{r_d}\frac{1}{x_{s-1}-1}.\end{aligned}$$

and thus by elementary manipulation and changing indices $x_i - 1 \longleftrightarrow x_i$, (1) results. ■

We can explicitly obtain the moments of the probability distribution analytically. To this end, we first obtain the moment generating function, and then use this to derive the mean and variance.

Lemma 3.3: The generating function $H(z)$ for $P(S = s)$ given from (1) is:

$$H(z) = \frac{zr_d!}{R!}(z + r_d)(z + r_d + 1)\ldots(z + R - 1) \qquad (2)$$

Proof: Note that (1) can be rewritten:

$$P(S=s) = \frac{r_d}{R}\sum_{r_d \leq x_1 < x_2 < \ldots < x_{s-1} \leq R-1}\frac{1}{x_1x_2\ldots x_{s-2}x_{s-1}}$$

for $s = 1, 2, \ldots, R - r_d + 1$.

Let $H(z)$ be the generating function for $P(S = s)$ in Lemma 3.2, namely,

$$H(z) = \sum_{s=1}^{R-r_d+1} P(S=s)z^s.$$

For ease of notation, we first use $G(z)$, the generating function for $P(S = s+1)$ rather than $H(z)$, the function for $P(S = s)$. Consider the function

$$G(z) = \frac{r_d}{R}(1 + \frac{1}{r_d}z)(1 + \frac{1}{r_d+1}z)\ldots(1 + \frac{1}{R-1}z).$$

which can be written,

$$G(z) = \frac{r_d!}{R!}(r_d + z)(r_d + 1 + z)\ldots(R - 1 + z). \tag{3}$$

To see that this is the generator function for $P(S = s + 1)$, one can multiply the terms of $G(z)$ which shows that the coefficient of z^{s-1} is

$$\frac{r_d}{R} \sum_{r_d \le x_1 < x_2 < \ldots < \; x_{k-1} \le R-1} \frac{1}{x_1 x_2 \ldots x_{k-2} x_{k-1}}.$$

Note that

$$H(z) = \sum_{s=1}^{R-r_d+1} P(S = s) z^s = z \sum_{s=1}^{R-r_d+1} P(S = s) z^{s-1} = zG(z).$$

An explicit expression for $H(z)$ is then, using (3), given by (2). ■

Lemma 3.4: Given the probability distribution (1), the expected number of probes required is

$$E(K) = 1 + \sum_{j=r_d+1}^{R} \frac{1}{j} \tag{4}$$

and the variance is:

$$Var(K) = \sum_{j=r_d+1}^{R} \frac{1}{j} - \sum_{j=r_d+1}^{R} \frac{1}{j^2}. \tag{5}$$

Proof: The mean and variance of probability distribution, given its generator function $H(z)$, are $H'(0)$ and $H''(1) - H'(1) - [H'(1)]^2$ respectively (e.g. [27]). From (2),

$$H'(z) = zG'(z) + G(z) \tag{6}$$

and

$$H''(z) = zG''(z) + G'(z) + G'(z) = zG''(z) + 2G'(z) \tag{7}$$

Now taking the logarithm of both sides of (3),

$$\log G(z) = \log \frac{r_d!}{R!} + \sum_{i=r_d}^{R-1} \log(i + z)$$

so that

$$G'(z) = G(z) \sum_{i=r_d}^{R-1} \frac{1}{z + i} \tag{8}$$

and

$$G''(z) = -G(z) \sum_{i=r_d}^{R-1} \frac{1}{(z + i)^2} + G'(z) \sum_{i=r_d}^{R-1} \frac{1}{z + i}. \tag{9}$$

From (8) and (9) we see that

$$G'(1) = G(1) \sum_{i=r_d}^{R-1} \frac{1}{1 + i} \tag{10}$$

and

$$G''(1) = -G(1) \sum_{i=r_d}^{R-1} \frac{1}{(1 + i)^2} + G'(1) \sum_{i=r_d}^{R-1} \frac{1}{1 + i} \tag{11}$$

Now using (3),

$$G(1) = \frac{r_d!}{R!}(r_d + 1)(r_d + 2)\ldots R = 1 \tag{12}$$

Using (6), (10) and (12), $H'(1) = G'(1) + G(1) = \sum_{i=r_d}^{R-1} \frac{1}{1+i} + 1$, from which (4) results. To obtain the variance, note that, given (7), (10), and (11),

$$H''(1) = G''(1) + 2G'(1) = -\sum_{j=r_d+1}^{R} \frac{1}{j^2} \left[\sum_{j=r_d+1}^{R} \frac{1}{j}\right]^2 + 2\sum_{j=r_d+1}^{R} \frac{1}{j}. \tag{13}$$

Using (13) and (4) we see that

$$\begin{aligned} Var(S) = -\sum_{j=r_d+1}^{R} \frac{1}{j^2} + \left[\sum_{j=r_d+1}^{R} \frac{1}{j}\right]^2 + 2\sum_{j=r_d+1}^{R} \frac{1}{j} + \\ 1 + \sum_{j=r_d+1}^{R} \frac{1}{j} - \left[1 + \sum_{j=r_d+1}^{R} \frac{1}{j}\right]^2 \end{aligned} \tag{14}$$

from which (5) results. ■

Lemma 3.5: For large R, and r_d fixed, the expected value is approximately $\log(R/r_d)$, and the variance approximately

$$Var(S) \approx \log(R/r_d) - \frac{\pi^2}{6} + \sum_{j=1}^{r_d} \frac{1}{j^2}. \tag{15}$$

Proof: Consider a fixed r_d and let $R \to \infty$. From (4),

$$E(S) = 1 + \sum_{j=r_d+1}^{R} \frac{1}{j} \approx 1 + \log(R/r_d) \approx \log(R/r_d).$$

Using (5) and the fact that $\sum_{j=r_d+1}^{\infty} \frac{1}{j^2} = \frac{\pi^2}{6}$ we find that

$$\begin{aligned} Var(S) &= \sum_{j=1}^{R} \frac{1}{j} - \sum_{j=1}^{r_d} \frac{1}{j} - \sum_{j=1}^{R} \frac{1}{j^2} + \sum_{j=1}^{r_d} \frac{1}{j^2} \\ &\approx \log(R/r_d) - \frac{\pi^2}{6} + \sum_{j=1}^{r_d} \frac{1}{j^2} \end{aligned}$$

as in (15). ■

Theorem 3.6: If replicas of a document neither arrive nor leave the system during a search, AWARE has a probability of finding the document in s steps according to (1). It thus also has an expected number of steps and variance as given in (4) and (5) respectively.

Proof: Nearest replicas are probed successively and replicas have been uniformly distributed among all possible replicas by the hash function $h(\cdot, \cdot)$. Thus, probing the nearest replica is equivalent to searching the replica space $1, \ldots, R$ randomly and thus we can directly apply Lemma 3.2. ■

D. Dynamic Behavior

We now consider the case when the replica situation alters during a search.

Theorem 3.7: If replicas of a document arrive or leave the system during a search, AWARE will perform as in Theorem 3.6, but is not guaranteed to return the closest document. It is guaranteed that the replica chosen for document download will be at least as close as the closest replica that persisted during the entire search.

Proof: The correctness and termination Lemmas (3.1 and 3.2, respectively) still apply under dynamic conditions. The only difference is that in a dynamic system, the addition of

Listing 4 K-PROBES: Location-unaware parallel probes

```
r ← R;
while $r \geq 1$ do
    $p \leftarrow \min(k, r)$; /* Number of probes this turn */
    $\mathcal{P} \leftarrow$ ($p$ distinct random indices from $[1, r]$);
    $\forall i \in \mathcal{P}$ : Check for document $h(i, d)$ in parallel;
    if any request was successful then
        return document retrieved from closest actual replica;
    end if
    $r \leftarrow \min(\forall i \in \mathcal{P}) - 1$;
end while
return nil;
```

replicas may cause the creation of a replica outside the current range. The effect is that freshly-added replicas may be better than the one actually selected for download. (In fact, only previously probed nodes that claimed non-existence at probe time but later turned into replica holders can be closer.) ■

E. Basic Lookup: Location-Unaware, Reliable Case

Not all DHTs support location awareness or can easily be equipped with a location service. In this case, *any* replica (not necessarily close) can be located by choosing probes differently. Care should be taken not to select a deterministic method, such as exactly halving the range. This would result in every request going to the same replica, negating the benefits of distributed document retrieval.

The recommended way is thus to select the next probe from the range using a uniformly distributed random process. This closely follows the location-aware algorithm, resulting in the same properties, such as efficiency and good distribution properties, with the notable exception of unknown distance.

If preference should be given to local servers, instead of a single probe, k probes can be sent in parallel, resembling a $(k+1)$-ary search. While this also may be useful in location-aware scenarios, the potential benefit seem to mainly lie in the location-unaware (i.e., random probing) scenarios.

To avoid taxing the system excessively, care should be taken not to directly issue a request for the entire document, but merely a probe for its existence, unless the document is known to be very small. Listing 4 illustrates a basic algorithm for k parallel probes. For ease of explanation, it assumes that a node that has replied and indicated having a replica will still have it in the short interval between probe response and document request. Section III-F presents a more sophisticated version.

F. Full Lookup

In the above descriptions, we have assumed the existence of a reliable network transport, which is often not the case. Therefore, the lookup function needs to be able to handle slow responses or machines that do not respond at all. Reasons include overload and outages of network links or nodes. Therefore, the algorithm needs to be able select a replica with fewer answers. The lack of an answer (or a "resource status unknown" reply issued during DHT reconfiguration) cannot be taken as an indication to shrink the probing range. Only true negative answers ("I do not have the document") can be used to shrink the range.

Listing 5 LOOKUP: Full algorithm; handles unresponsive nodes and timeouts

```
$r \leftarrow R$;
$\mathcal{B} \leftarrow \emptyset$; /* Blacklist of unresponsive nodes */
label retry;
while $r \geq 1$ do
    $b \leftarrow \min(k, |[1, r] \setminus \mathcal{B}|)$; /* Number of probes */
    $\mathcal{P} \leftarrow$ ($b$ distinct indices from $[1, r] \setminus \mathcal{B}$); /* Pick according to distance metric or randomly */
    $\forall i \in \mathcal{P}$ : Send query for document $h(i, d)$;
    Start timeout with period $\tau$;
    while fewer than $\min(b, q)$ replies processed this turn do
        Wait for timeout or next reply;
        if timeout then
            $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{P}$;
            goto retry;
        end if
        $Y \leftarrow$ instance index of replying node;
        if reply was positive then
            if document retrieval successful then
                return document;
            end if
        else
            $r \leftarrow \min(r, Y - 1)$; /* Never raise r again */
        end if
    end while
end while
return nil;
```

Listing 5 describes the full lookup algorithm that also handles timeouts, unreachable nodes, and nodes ceasing to be replica holders during the query process. The behavior is controlled by τ, the per-step timeout period, and q, the number of query replies that are sufficient before continuing to the next step.

The algorithm is able to take advantage of locality, but will also work without that information. (The generalization to use a sliding window of at most k outstanding messages that have not yet timed out is straightforward.) It successfully returns a document if at least one of the replica holders was reachable (within the timeout constraints) for the entire duration of the algorithm. In general, this means that if at least one replica's holder is reachable, the replica will be found. Of course, with an increasing number of unreachable nodes, an increasing effort is required to find this document. Nevertheless, this property makes RE useful even in harsh environments such as ad-hoc networks.

IV. MEASUREMENTS

We compared, by simulation, the effect on total time to obtain a replica, when location information is used to choose the nearest replica (AWARE algorithm) and when a node is picked randomly independent of cost (random algorithm). To obtain realistic delay information, we estimated the delay distribution from data obtained from [28], which consisted of RTT probes from 5 locations (3 different ISPs in Switzerland, one in Japan and one in the United States).

All results are obtained from the average of 500 simulation runs, including error bounds at 95% confidence. We show sample confidence intervals whenever they do not detract from visibility. In all cases, the intervals were verified to be

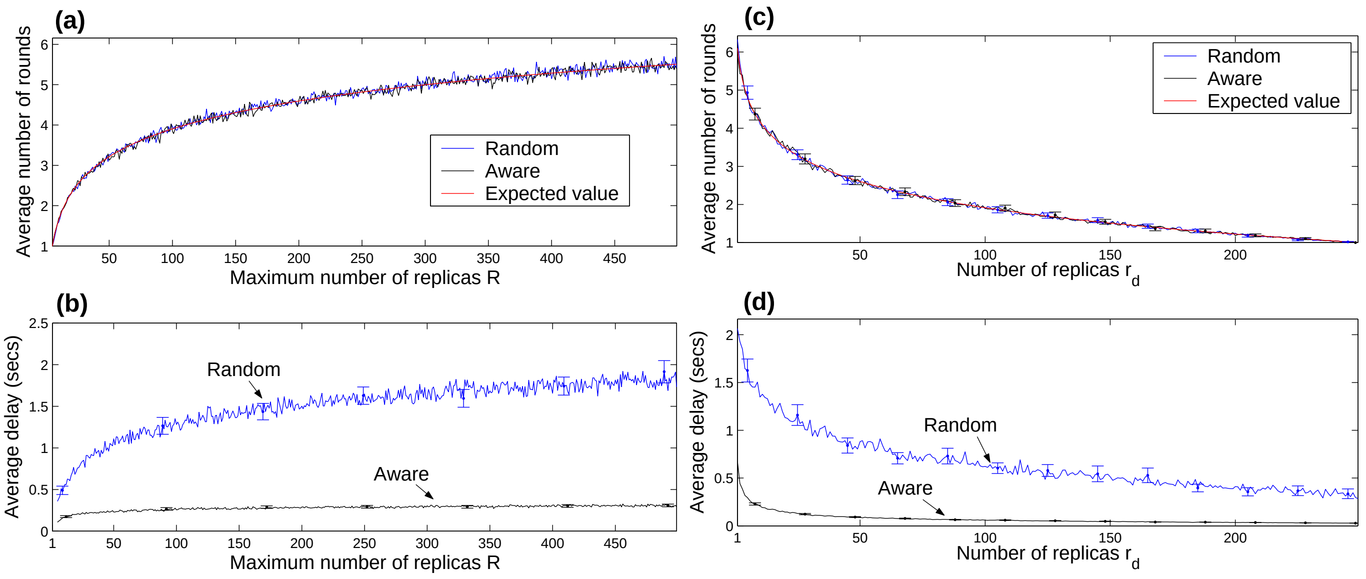

Fig. 4. (a) The average number of probing rounds needed as a function of the maximum number of replicas R (for $r_d = 5$ replicas of the document) when using the AWARE and random algorithms both agree with the expected value from (4). (b) Using AWARE, the average time to obtain the document does not, in general, increase as a function of R. (c) The average number of probing rounds needed as a function of the number of replicas r_d that exist (for $R = 250$ maximum replicas) when using the AWARE and random algorithms both agree with the expected value from (4). (d) Using AWARE, the average time to obtain the document decreases significantly per additional replica when for relatively few replicas, but the benefit is minimal after a small critical mass.

in agreement with the general conclusions. The results are illustrated in Figure 4. The experimental results agree with the derivations in Section III-B, and show that AWARE is far quicker at retrieving the document than just location unaware probing.

Interestingly (Figure 4(b)), the average time to obtain the document does not, in general, increase as the maximum number of replicas R increases. This confirms the assertion that even choosing R to be large has little negative influence on the speed of retrieving a document. In addition, the relative decrease in search time and number of rounds by adding further replicas (i.e. increasing r_d) is small after the initial boost of adding a few replicas (Figure 4(d)). This property of an initial boost is related to the oft remarked "power of two choices" [29].

V. Applications

RE opens up new possibilities for applications that require high-performance access to dynamically replicated and up-to-date information content. It enables a wide variety of applications, ranging from a *uniform paradigm* for web serving/web mirroring/content distribution/web caching to distributed storage or collaboration in ad-hoc wireless networks. The latter environment differs significantly from wired networks in terms of link and node reliability as well as latency and transmission costs. Reliable location of a close-by replica is thus of key importance. Constant probing by replica holders to ascertain whether they still are up to date would also incur unacceptable loads on the ad-hoc network.

RE, is well suited for many applications and those with specific requirements can use RE in combination with more traditional approaches such as caching, redirection, centralized directories, or distributed computation, as described below.

Caching: Even though RE often eliminates the need for caching, there may be instances where additional caches can be useful, including disconnected operation. In fact, RE improves the scalability of replication, compared with a single original document, because verifying whether cache contents are still current (or installing an update callback with a replica holder) imposes additional load on the replica holder. Having more up-to-date replica holders available will significantly reduce the per-replica-holder load.

Redirection: When a particular replica holder is overloaded, it may return a list of pointers to known caches or other nearby replicas (or replica holders) instead of incurring additional load by returning the document.

Directory Backup: RE may also serve as a distributed backup solution to centralized replication directories. If the directory places at least some replicas per document according to the RE rules, the system will continue to work even if the directory service becomes unavailable. The directory server may have placed some of the replicas at strategic locations independent of the RE rules to obtain better performance or physical security according to specialized rules. These replicas will not be used during the outage of the directory server, but all others will continue to work as if nothing happened: the perfect failsafe solution.

Distributed Computation: The resources addressed through DHT and RE need not necessarily be data blocks, but they can also be programs. RE can thus be used as a platform for distributed computation.

VI. Summary and Conclusions

Distributed storage systems based on DHTs promise the provision of reliable and resilient access to data, a key element of which is a powerful replica management system. We have identified a catalog of criteria that can be used to classify these systems and then described a novel method, *replica enumeration*, that fulfills all of the criteria identified with the exception of location variability (Table I).

In summary, RE is a fully distributed approach that requires neither state nor control information, while providing a very efficient lookup performance. Its power and versatility make RE very useful in a wide range of systems, ranging from server-farm load balancing over a distributed backup solution for centralized directory systems to scalable globally distributed storage systems.

Our simulations show that even locality-unaware (random probing) systems perform well and that the expected number of probing rounds is of logarithmic order This indicates that it scales well with the maximum number of replicas, R, and that there is already a significant performance improvement for popular documents (i.e., replicated on more than one node) with only a few replicas.

Using locality information considerably improves on the document retrieval delay. The expected time to locate a document is, to all and intents and purposes, independent of R and very small: only a couple of typical Internet round-trip times.

The next steps include examining further applications, derivation of hash functions with locality properties, evaluation of approaches to improve the load-balancing properties without additional latency, and extension of the quantitative analysis to the location-unaware and K-PROBE cases.

References

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, USA, Aug. 2001, pp. 149–160.

[2] G. N. Frederickson, "Searching intervals and compact routing tables," *Algorithmica*, vol. 15, no. 5, pp. 448–466, May 1996.

[3] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "SkipNet: A scalable overlay network with practical locality properties," in *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Mar. 2003.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of ACM SIGCOMM*, Sept. 2001.

[5] M. Waldvogel and R. Rinaldi, "Efficient topology-aware overlay network," *ACM Computer Communications Review*, vol. 33, no. 1, pp. 101–106, Jan. 2003, proceedings of ACM HotNets-I (October 2002).

[6] K. Aberer, M. Hauswirth, M. Punceva, and R. Schmidt, "Improving data access in P2P systems," *IEEE Internet Computing*, vol. 6, no. 1, Jan./Feb. 2002.

[7] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov. 2001, pp. 329–350.

[8] B. Y. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," University of California, Berkeley, Tech. Rep. UCB/CSD-01-1141, April 2001.

[9] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Exploiting network proximity in distributed hash tables," in *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, O. Babaoglu, K. Birman, and K. Marzullo, Eds., June 2002, pp. 52–55.

[10] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of DNS-based server selection," in *Proceedings of IEEE INFOCOM*, Anchorage 2001, pp. 1801–1810. [Online]. Available: citeseer.nj.nec.com/article/shaikh01effectiveness.html

[11] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," in *Proceedings of ACM SIGCOMM*, Sept. 1998, pp. 254–265.

[12] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–59, September-October 2002.

[13] S. C. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in *Proceedings of INFOCOM 2002*, 2002. [Online]. Available: citeseer.nj.nec.com/rhea02probabilistic.html

[14] V. Ramasubramanian and E. G. Sirer, "Beehive: Exploiting power law query distributions for o(1) lookup performance in peer to peer overlays," in *First Symposium on Networked Systems Design and Implementation (NSDI)*, Mar. 2004.

[15] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *Symposium on Operating Systems Principles*, Oct. 2001, pp. 202–215.

[16] M. Leslie, J. Davies, and T. Huffman, "A comparison of replication strategies for reliable decentralised storage," *Journal of Networks*, vol. 1, no. 6, pp. 36–44, Nov./Dec. 2006.

[17] M. Waldvogel, P. Hurley, and D. Bauer, "Dynamic replica management in distributed hash tables," IBM, Research Report RZ–3502, July 2003.

[18] C. Harvesf and D. M. Blough, "The effect of replica placement on routing robustness in distributed hash tables," in *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, 2006, pp. 57–6.

[19] D. Katabi and J. Wroclawski, "A framework for scalable global IP-anycast (GIA)," in *SIGCOMM*, 2000, pp. 3–15. [Online]. Available: citeseer.nj.nec.com/katabi00framework.html

[20] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *ACM SIGCOMM 2002*, Aug. 2002. [Online]. Available: citeseer.nj.nec.com/cohen02replication.html

[21] B.-G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. H. Papadimitriou, and J. Kubiatowicz, "Selfish caching in distributed systems: A game-theoretic analysis," in *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing*, July 2004.

[22] D. Tsoumakos and N. Roussopoulos, "APRE: A replication method for unstructured P2P networks," University of Maryland, Technical Report CS–TR–4817, Feb. 2006.

[23] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proceedings of IEEE INFOCOM*, New York, NY, USA, June 2002, pp. 170–179.

[24] L. Garcés-Erice, K. W. Ross, E. W. Biersack, P. A. Felber, and G. Urvoy-Keller, "TOPLUS: Topology-centric lookup service," in *Proceedings of Networked Group Communications (NGC) 2003*, Sept. 2003.

[25] 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 *ACM Symposium on Theory of Computing*, May 1997, pp. 654–663. [Online]. Available: citeseer.ist.psu.edu/karger97consistent.html

[26] R. M. Karp, "Probabilistic recurrence relations," in *Proceedings of the twenty-third annual ACM symposium on Theory of computing*. ACM Press, 1991, pp. 190–197.

[27] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1994.

[28] R. Rinaldi and M. Waldvogel, "Routing and data location in overlay peer-to-peer networks," IBM, Research Report RZ-3433, July 2002.

[29] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, 2001.