A Case for Unstructured Distributed Hash Tables

Krishna P. N. Puttaswamy and Ben Y. Zhao
Computer Science Department, U. C. Santa Barbara
{*krishnap, ravenben*}@*cs.ucsb.edu*

***Abstract*—Structured peer-to-peer overlays support compelling applications such as large-scale file systems and distributed backup using the distributed hash table (DHT) interface. While unstructured file-sharing systems continue to flourish, wide adoption of structured applications has been elusive. We explore an alternative path to deployment of these applications by asking the question, can structured applications be run on top of unstructured overlays? We build an unstructured distributed hash table (UDHT) on top of state of the art search and topology management mechanisms, and evaluate whether it can sufficiently emulate properties of DHTs to support structured applications.**

I. Introduction

Research on structured peer-to-peer overlay networks [21], [20], [18] has produced a number of compelling large-scale distributed applications, everything from Internet-scale storage [17], distributed backup [4] to application-level multicast [2] and cooperative web caching [13]. Despite their obvious advantages, these applications have seen relatively limited deployment in the wild, with the only exceptions being file-sharing networks [7] and infrastructure-based services such as Coral [8].

The exact cause for the limited deployment of structured overlay applications is unclear. Some believe that currently proposed applications are too resource-intensive for the typical home user. Others argue that structured overlays are too complex, incur too much control overhead, or are not robust against node churn. These arguments have been disputed by both recent literature [1], [16] and by the successful deployment of DHT-based file-sharing networks such as E-donkey [7]. Finally, since most mechanisms on structured overlays rely on probabilistic algorithms, *e.g.* data placement/location and load-balancing, many believe that at scales below a critical threshold, *e.g.* thousands of users, structured applications may deviate from expected performance, exhibiting imbalance in load distribution and incomplete and inconsistent routing table entries. Similarly, their advantages of reduced per-node routing state (logarithmic to size of network) only become evident at large scales. This provides an "inverse scaling" problem, where smaller structured overlay networks exhibit potentially undesirable properties, thus impeding the growth of structured applications to larger user populations.

In this paper, we attempt to circumvent these challenges through an alternative deployment path for structured overlay applications. As a start, we explore the feasibility of efficiently implementing the basic features of a DHT on unstructured overlays and thus deploying DHT-based structured overlay applications on an Unstructured Distributed Hash Table (UDHT). Supporting structured applications on unstructured overlays means we could avoid pitfalls such as the inverse scaling problem and leverage deployed file-sharing networks to bootstrap a large-scale structured network. In addition, our UDHT would support flexible searches provided by file-sharing systems along with location of rare objects like DHTs. Unlike hybrid networks [14], a UDHT supports both unstructured and structured overlay functionality in a single network.

The key contributions of this paper are: One, we leverage state-of-the-art algorithms in data location, data replication, and topology management from unstructured P2P networks, and synthesize the core data placement and data location algorithms of UDHT. Two, we propose support mechanisms for data maintenance and replication, providing a fully functional implementation of the DHT interface commonly used by structured P2P applications (Section III). Three, we measure the effectiveness of UDHT, through detailed simulation, to locate *any* randomly chosen file and thus provide the deterministic data storage and retrieval functionality required by structured applications (Section IV). Though we sacrifice the guaranteed lookup properties provided by the DHTs, our evaluation shows promising results to motivate further investigation in this direction. Four, We identify the main hurdles in making our UDHT approach practical to support large-scale structured applications and present (Section V) a discussion of further optimizations to improve UDHT.

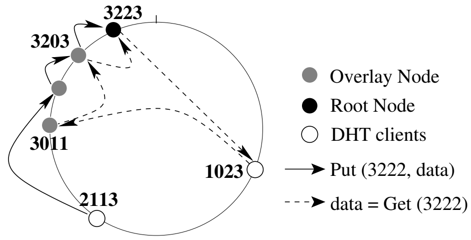

Fig. 1. *Distributed Hash Table.* Node `2113` stores data into the DHT using a put operation with key `3222`. Node `1023` retrieves the data using get of key `3222`.

II. Background and Related Work

Before discussing our system design, we introduce the basic concepts behind structured peer-to-peer overlays and highlight several key pieces of related work.

Structured Overlays and DHTs. A structured peer-to-peer overlay is an application-level network connecting any number of nodes, each representing an instance of an overlay participant. Nodes are assigned nodeIds uniformly at random from a large identifier space. Application-specific objects are assigned unique identifiers called keys from the same space.

The overlay dynamically maps each key to a unique live node, called its *root node*. While a key's root can change with network membership, at any given time in a consistent network, a single node is responsible for each key. The root is usually defined as the peer with nodeId closest to the key. At their lowest layer, structured overlays provide *key-based routing* (KBR) [6], delivering messages based on a destination key to the key's root node using multihop routing, where a node at each hop forwards the message using a local routing table of overlay links.

While KBR provides the foundation for structured overlays, the most commonly used interface by structured applications is the DHT interface [5], [6]. A DHT provides reliable storage by storing each data block or file based on a specific "key." It provides two simple functions, *put (Key K, Data D)* reliably stores the data accordingly to key K, and *get (Key K)* retrieves the data stored associated with key K. Finally, a DHT maintains high availability of its data across changes in network membership. Figure 1 shows a DHT operation, where node `1023` retrieves data stored by node `2113`.

Related Work. A number of previous overlay network studies have strongly influenced our work. First, two studies by Gkantsidis et al. [10], [11] analyzed the performance of random walk search algorithms on overlay networks, and showed them to be highly effective when coupled with one-hop replication. Lv et al. evaluated different search algorithms on unstructured overlays, and showed that k-random walks perform significantly better than flooding or expanding ring searches [15]. Gia [3] exploits the heterogeneous capacity across peers for topology adaptation, resulting in the natural selection of high capacity nodes as network hubs, greatly increasing the effectiveness of biased random walk searches. Our work leverages Gia's capacity-based connectivity model. Recent work showed that structured overlays are capable of supporting complex queries using similar adaptivity and robustness optimizations as their unstructured counterparts [1]. In comparison, our work asks the opposite question: can unstructured overlays provide structured overlay interfaces such as a DHT.

Finally, our experiments leverage datasets from several peer-to-peer measurement studies[1]. We use the original Gnutella study [19] to derive a Gnutella network topology, as well as a trace for node churn. We also follow the Gia [3] and Myths [1] work in deriving a heterogeneity capacity model from the Gnutella data. From recent measurement studies on the E-Donkey network [7], we derive a model for file distribution. Finally, we also utilize the node churn trace from the recent Cornell study on the Skype network [12].

III. Searching in Unstructured DHTs

Despite recent studies that have dispelled some of the criticisms of structured overlays, the limited deployment of structured applications remains unexplained. Whether the challenges are based on complexity, inverse scaling, or unknown issues with structured overlays, one potential solution is to decouple structured overlay applications from structured overlays and leverage existing deployments of unstructured networks. Using this approach, we build and deploy an Unstructured Distributed Hash Table (UDHT) that provides the same interface as a traditional DHT. We can bootstrap structured applications on a smaller unstructured network running a UDHT, then gradually migrate to structured network at larger scales. The focus of this paper, is to evaluate the feasibility of building a UDHT to support structured overlay applications at smaller scales. Issues of migration to structured overlays remain future work.

While unstructured overlays have successfully supported queries on file-sharing networks, they are optimized for locating popular files with a high number of replicas in the network. The success of our approach relies on the ability of unstructured overlays to efficiently

[1]We gratefully acknowledge our appreciation to the authors of these studies for access to their measurement data

locate all files, including those with a low replication factor. We begin by explaining our choice of search algorithms and topologies, then describe our implementation of an unstructured DHT network.

A. Search and Topology Management

Search. Previous work [3], [10], [11], [15] has studied a number of search algorithms in unstructured overlays, including TTL-based flooding, expanding ring searches, and k-random walks. In TTL-based flooding, a node floods a query to all of its neighbors, decrementing the TTL count with each hop. Queries are dropped when TTL reaches zero. Expanding ring searches extend TTL-based flooding by repeatedly querying with increasing TTL values until the object is found. Finally, a random walk query randomly traverses the network graph until it finds the object or a maximum hop count is reached. A k-random walk issues k random walk queries in parallel.

Several studies [3], [15] show that k-random walks significantly outperform flooding-based approaches. We evaluate different variants of random walk algorithms for our system. However, where Gia's biased random walk requires per-query state at each node to avoid redundant paths, we introduce an *embedded n-window* random walk, where the last n nodes visited are embedded inside each query. A query avoids any neighbors already present in its n hop window, thus avoiding any routing loops of n hops or less. If the query reaches a "dead-end" where all neighbors are in the n-hop window, it chooses the neighbor least recently traversed.

Topology Management. To improve search performance, we require the unstructured overlay to exploit heterogeneous node capacity using the same technique described in Gia [3] and Myths [1]. A node's maximum connectivity is roughly proportional to its "capacity," which is modeled using its bandwidth capacity. Associating degree with node capacity produces networks with heterogeneous per-node connectivity, where a large number of less connected "edge" nodes are connected by a number of highly connected "hubs." Since a node's chance of being visited by a query is proportional to its in-degree, random walk queries are more likely to visit hub nodes than edge nodes.

One-hop Index Replication. Finally, we adopt the one-hop index replication optimization introduced by Gia [3], where each node maintains an index of objects stored by its one-hop neighbors. Combined with the capacity-based connectivity model, this significantly improves search effectiveness by caching indices of edge nodes on hubs often visited by random walks. This emulates multi-layer superpeer search, and explains how Gia outperforms a two-layer superpeer network.

B. Implementing a UDHT

Supporting Put and Get. A DHT supports two basic operations: `PUT` stores a data object based on an identifying key K, and `GET` retrieves the data object associated with K. To support additional DHT semantics such as authentication, data removal, and user quotas, objects stored using the same key must be co-located on the same set of peers.

The UDHT maintains n replicas of each object for increased availability, where replication factor n is a system-wide parameter determined at startup time. Unlike a DHT, K's replicas can be on any n peers, and is independent of K.

To perform a `GET` operation on K, a client initiates a k-random walk search for key K. The random walks continue until each has successfully located the desired object, or has reached its `MAXCOUNT` number of hops. To perform a `PUT` on K, a client first searches for any existing objects stored with key K by performing a `GET` on K. If such objects exist, it is highly likely the search will locate at least one peer P storing a replica. If no replica is found, the client issues k parallel random walks to choose a set of n nodes to store the replicas of K. At each new node, a random walk terminates or adds the node to the set with probability $1/m$, where m is an approximate estimate of the network diameter. Once n peers are found, the client disseminates the object to all peers in the set.

Data Replication and Proactive Maintenance. We assume that once a new node connects to the UDHT, it maintains connections to its overlay neighbors until it leaves the UDHT. Each peer sends periodic heartbeats to its one-hop neighbors. When a peer notices a neighbor X has disconnected, it reads its own list for objects stored on X (from one-hop index replication), performs a random walk search for each object and makes another copy of the object on itself.

IV. Experimental Evaluation

In this section, we perform detailed evaluation of our mechanisms and algorithms using a customized simulator based on PlanetSim [9], a Java-based event simulator for overlays. For realistic results, we leverage a variety of data from previous measurement studies [3], [7], [12], [19], including overlay topology, node capacity, and overlay churn models. Note that we are primarily concerned about performance at the overlay layer, *e.g.*

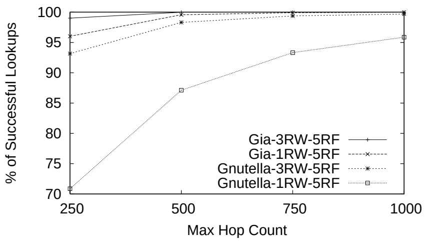

Fig. 2. Query success rate with one-hop replication on the Gnutella and Gia topology.

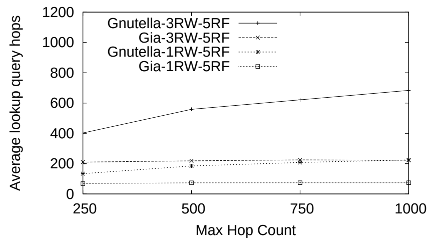

Fig. 3. Query overhead with one-hop replication on the Gnutella and Gia topology.

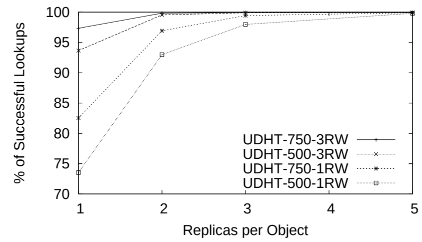

Fig. 4. Query success rate with UDHT on a static network.

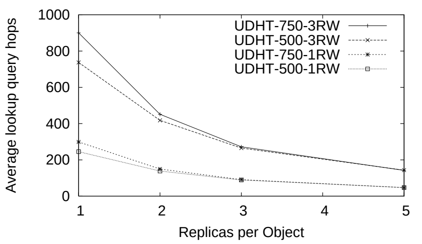

Fig. 5. Query overhead with UDHT on a static network.

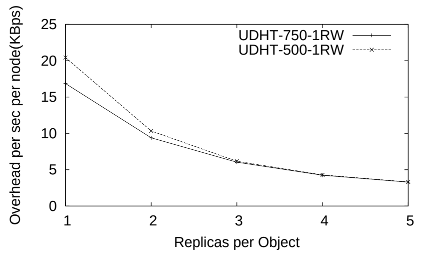

Fig. 6. Bandwidth consumed in a 3K node UDHT, 1 query (68Bytes) per sec/node.

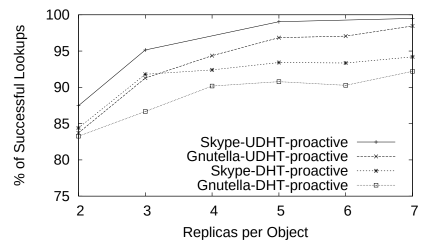

Fig. 7. Comparison of query success rate of 3K DHT and UDHT (1-RW, 750 hops depth) under different churns.

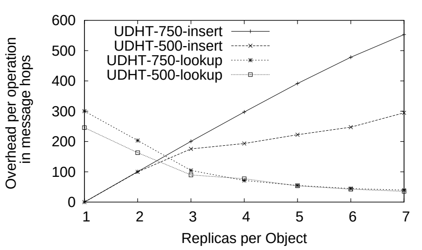

Fig. 8. Relative overheads of queries and object placement in UDHT.

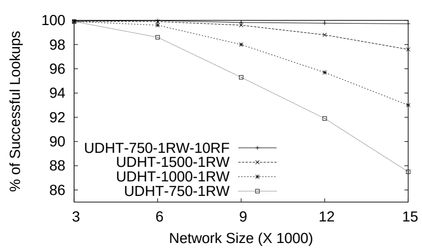

Fig. 9. Comparing query success with varying network sizes for UDHT, 1RW, 5RF.

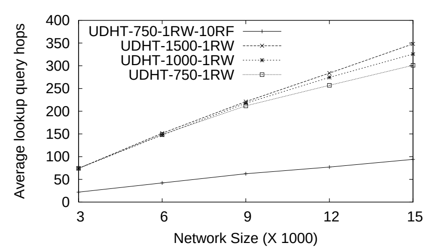

Fig. 10. Comparing query overhead with varying network sizes for UDHT, 1RW, 5RF.

overlay hops per query, and make no assumptions about the topology-awareness of the overlay network.

We perform two sets of experiments. First, we evaluate the ability of existing topology management and search algorithms to locate "unpopular" objects on two overlay topologies, Gnutella and Gia. Second, we evaluate the effectiveness of our UDHT by examining query success under stable, Gnutella, and Skype churn models.

A. Evaluation Methodology

Overlay Topology. We evaluate our system on two overlay topologies. First, we take topology data gathered by the first Gnutella measurement [19], and isolate its largest connected component (1787 nodes) for our "Gnutella" topology. Second, we produce a 3000 node "Gia" topology based on the capacity-based adaptation parameters from Gia [3]. As in Gia, we assign node capacities by mapping Gnutella bandwidth measurements to capacity values, then connect the network until nodes reach a "satisfied" state.

Overlay Churn Traces. For evaluation under network dynamics, we use two traces of churn. First, we generate a list of node lifetime values from application-level uptime information of 37K Gnutella hosts [19]. For each new node, we assign a randomly selected lifetime value from this distribution. Second, we use similar techniques to model the lifetime values of more than 2100 Skype superpeers from a recent Cornell study [12]. In both cases, we keep the network size constant by adding a new node whenever an existing node leaves the network. New nodes in the "Gnutella" model are assigned a node degree from the Gnutella connectivity distribution, and nodes from the "Gia" model are assigned the same capacity as the node it replaced. Neighbors of the new node are chosen randomly while respecting connectivity or capacity constraints at each node.

Object Replication and Placement. Since our intent is to study search for "unpopular" objects, each object in our experiments has a replication factor of 5 unless otherwise specified. For Figures 2 and 3, we randomly map each new node into one of 12000 nodes from a recent study of the e-donkey network [7], thereby assigning it a selection of object replicas. For UDHT experiments, each new node brings in 100 new objects that are then replicated into the system using the `PUT` operation. Replica placement also observes each node's capacity based on the Gia distribution.

B. Evaluation Results

Our simulations look at several scenarios. First, we evaluate whether unstructured overlays can support effective location of "unpopular" objects, where the number of replicas is low (≤ 5). These results determine the parameters and algorithms for our UDHT. Second, we examine the performance of our UDHT under static and dynamic network conditions. We seek to understand our search overhead and the impact of increasing object replication. Third, we look at the control overhead for inserting objects into the network. Finally, we evaluate our UDHT performance as the network grows in size.

Queries on Static Overlays. Our preliminary results confirm previous studies [10], [11], [3] that show random walks to be the most effective search algorithms on unstructured overlays. We begin by evaluating the effectiveness of random walk strategies on both the Gnutella and Gia topologies. We quantify overhead using the number of overlay hops crossed by all messages.

Figures 2 and 3 show the lookup query success for different random walk depths and different number of random walks, for the Gnutella and Gia topologies. Overall, the capacity-based Gia topology provides significantly better query performance with lower overhead compared to the flat Gnutella topology. The variance in node degree, combined with one-hop replication allows random walks to cover a greater portion of the network. While using parallel random walks, *i.e. 1RW to 3RW*, improves query success rate, our experiments confirm prior work [3] that increasing the replication factor can more dramatically improve query success rate with lower overhead. From these results, we conclude that our UDHT will enable one-hop index replication, use only 1 random walk for search, and provide object replication. Where possible, we will utilize a capacity-based topology to improve query success.

UDHT Under Static Conditions. Figures 4 and 5 evaluate UDHT query success rates and search overheads on static networks. Figure 6 presents the bandwidth required by each node if every node maintained a constant stream of queries at a rate of 1 query/second, with average query size of 68 Bytes. Our UDHT has one-hop index replication enabled, and is running on a 3000 node Gia topology. Clearly, increasing the object replication factor drastically improves query performance and reduces search overhead. While performing 3-random walk drastically increases search overhead, its benefit to queries is limited and easily offset with slightly higher object replication. In addition, increasing the random walk depth improves query success with a very low increase in query cost. Therefore, unless otherwise specified, we will use 1-random walk with a max hop count of 750 for our remaining UDHT experiments.

UDHT and DHT Under Network Churn. Figure 7 presents a comparison of the lookup performance of a 3000 node DHT and UDHT network under the Skype and the Gnutella churn models. We used Chord for our DHT tests and, for a fair comparison, we used highly favorable values for the stabilization interval and the finger table update interval. The UDHT network was configured to use 1-Random walk with 750 hops maximum depth. In general, the Skype model exhibits lower churn rate than the Gnutella model. Each test ran for approximately 12 hours of virtual time. During each run, an average of 840 nodes died under the Skype churn model, while an average of 2600 nodes died under the Gnutella model.

Our UDHT results show that proactive replica maintenance provides excellent object availability, leading to high query success even under churn. In our repair scheme, the neighbor of a failed node uses its neighbor index to search for the failed node's objects. We see from the graph that for networks with higher rates of expected churn, *e.g.* the Gnutella model, increasing object replication can dramatically improve data availability. Our experiments confirm that the overhead of each proactive recovery request is similar to a normal UDHT lookup and increases with higher churn, as expected. Under churn, even DHT lookup performance drops significantly and is comparable to the performance of a UDHT. Because of heavy churn, higher replication and proactive replica maintenance is required in a DHT network also to sustain high lookup performance. DHT results, again, show similar trend as the UDHT results with better lookup performance under Skype churn model than under Gnutella churn.

These results together indicate that with the right set of parameters and with proactive replication, its possible to achieve high data availability, similar data availability as that of a DHT, in an unstructured network.

Object placement overhead. We compare the relative costs of performing object placement (`PUT`), and query operations in our UDHT. Figure 8 shows that as the replication factor increases, the cost of placing data outpaces the cost of locating objects. Objects are easier to find, but finding random nodes for its replica set with enough capacity is more costly. However, each node can maintain a cache of local neighbor nodes by doing a random walk once and use the cache to speedup the puts, effectively reducing the object placement overhead to a one time cost.

Performance in larger networks. Figures 9 and 10 show the effect of increasing the network size on our UDHT algorithms. As network size increases, a given random walk slowly becomes less effective at locating data. To maintain a success rate, we can increase the random walk depth or the replication factor. Increasing the random walk depth increases the query overhead, while a higher replication factor decreases the query overhead and slows down the decrease in query success rate. We discuss further mechanisms to improve scalability in Section V.

Overall, our results show that with proactive object replication and maintenance, our UDHT can support the DHT interface on unstructured networks. However, sustaining good performance of UDHT under larger network sizes is a challenge we need to overcome to efficiently run structured applications on UDHT.

V. Discussion and Conclusions

Scaling UDHT. Figures 9 and 10 bring forth the need for additional mechanisms to scale UDHT to larger networks. We plan to investigate further optimizations to address this issue.

Related works [3], [10], [11] show the significant effect of 1-hop replication on UDHT lookup. In addition to index replication with 1-hop local neighbors, each node can perform 1-hop replication with a set of randomly selected long-distance neighbors. We conjecture that this spreading of indices to a larger area of the network should significantly decrease lookup failures in UDHT at low random walk depths. In addition to replicating, searching with k-random walks, with k *different* long-distance neighbor as origins, instead of a single origin, explores different neighborhoods of the network and can improve the query performance further. Selecting this set of long-distance neighbors, however, is a challenging problem that we are looking into.

In summary, we propose an Unstructured Distributed Hash Table to provide a DHT interface on unstructured P2P networks. Not only does the UDHT support location of rare objects through the DHT interface, it does not rely on structure, and thus also supports complex queries like unstructured file-sharing systems. Our results show that our data search and replication mechanisms are very effective in smaller networks. We are currently exploring a very promising technique to scale UDHTs to extremely large networks, and building software "plug-ins" to enable gradual conversion of file-sharing networks into legitimate end-user applications.

Acknowledgements

The authors gratefully acknowledge the support from DARPA through the Control Plane program (BAA04-11) and NSF through CAREER Award #0546216.

References

[1] Castro, M., Costa, M., and Rowstron, A. Debunking some myths about structured and unstructured overlays. In *Proc. of NSDI* (Boston, MA, May 2005).

[2] Castro, M., et al. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE JSAC 20*, 8 (2002).

[3] Chawathe, Y., et al. Making gnutella-like p2p systems scalable. In *Proc. of SIGCOMM* (August 2003).

[4] Cox, L. P., Murray, C. D., and Noble, B. D. Pastiche: Making backup cheap and easy. In *Proc. of OSDI* (Dec. 2002).

[5] Dabek, F., et al. Wide-area cooperative storage with CFS. In *Proc. of SOSP* (Banff, Canada, October 2001).

[6] Dabek, F., et al. Towards a common API for structured P2P overlays. In *Proc. of IPTPS* (Berkeley, CA, February 2003).

[7] Fessant, F. L., et al. Clustering in peer-to-peer file sharing workloads. In *Proc. of IPTPS* (San Diego, CA, February 2004).

[8] Freedman, M. J., Freudenthal, E., and Mazies, D. Democratizing content publication with coral. In *Proc. of NSDI* (San Francisco, CA, December 2004).

[9] Garcia, P., et al. Planetsim: A new overlay network simulation framework. In *Proc. of ASE* (Sept. 2004).

[10] Gkantsidis, C., Mihail, M., and Saberi, A. Random walks in peer-to-peer networks. In *Proc. of INFOCOM* (2004).

[11] Gkantsidis, C., Mihail, M., and Saberi, A. Hybrid search schemes for unstructured peer-to-peer networks. In *Proc. of INFOCOM* (Miami, FL, March 2005).

[12] Guha, S., Daswani, N., and Jain, R. An experimental study of the skype peer-to-peer voip system. In *Proc. of IPTPS* (San Diego, CA, February 2004).

[13] Iyer, S., Rowstron, A., and Druschel, P. Squirrel: A decentralized peer-to-peer web cache. In *Proc. of PODC* (2002).

[14] Loo, B. T., et al. Enhancing p2p file-sharing with an internet-scale query processor. In *Proc. of VLDB* (2004).

[15] Lv, Q., et al. Search and replication in unstructured peer-to-peer networks. In *Proc. of Supercomputing* (June 2002).

[16] Qiao, Y., and Bustamante, F. E. Structured and unstructured overlays under the microscope. In *USENIX* (2006).

[17] Rhea, S., et al. Pond: The OceanStore prototype. In *Proc. of FAST* (San Francisco, CA, April 2003).

[18] Rowstron, A., and Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware* (November 2001).

[19] Saroiu, S., Gummadi, P. K., and Gribble, S. A measurement study of peer-to-peer file sharing systems. In *Proc. of MMCN* (January 2002).

[20] Stoica, I., et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM* (2001).

[21] Zhao, B. Y., et al. Tapestry: A global-scale overlay for rapid service deployment. *IEEE JSAC 22*, 1 (January 2004).