Toward Network Data Independence

Joseph M. Hellerstein

EECS Computer Science Division, UC Berkeley Intel Research, Berkeley

Abstract

A number of researchers have become interested in the design of global-scale networked systems and applications. Our thesis here is that the database community's principles and technologies have an important role to play in the design of these systems. The point of departure is at the roots of database research: we generalize Codd's notion of data independence to physical environments beyond storage systems. We note analogies between the development of database indexes and the new generation of structured peer-to-peer networks. We illustrate the emergence of data independence in networks by surveying a number of recent network facilities and applications, seen through a database lens. We present a sampling of database query processing techniques that can contribute in this arena, and discuss methods for adoption of these technologies.

1 Introduction

A number of research communities are now investigating the design of global-scale networked systems made up of a very large, relatively volatile mix of participating compute resources. This work falls under various rubrics, including peer-to-peer (p2p) systems, overlay networks, and grid computing.

These efforts have strong echoes of the relational database revolution of the early 1970's, as we describe below. We believe that the database research community should be interested in this research agenda, and well-positioned to make fundamental contributions. The research is moving very quickly, and some key building blocks are emerging. We believe that collaborations across areas like databases, networking, and distributed algorithms could help shape a new generation of important technologies that could eventually move into – or become – the core of the Internet.

The relevance of database technology in this arena can be seen by viewing recent developments in traditional database terms. We survey a number of related efforts in both network applications and network infrastructure through a "database lens", and we highlight some database technologies that we are investigating in these contexts.

2 Data Independence Redux

The success of modern database systems is due in large part to Codd's notion of *data independence*. In traditional terms, data independence is the decoupling of a database system's application-level interface from its data organization. Data independence permits changes to be made to the layout of data (file organizations and indexes) without any modification to applications that access that data.[1]

The concept of data independence and its instantiation in the relational model has been heraled as being "as clear a paradigm shift as we can hope to find in computer science" [21], and earned Codd his Turing award. Yet the message of data independence has had limited impact outside the database community. We suspect that data independence – properly generalized – has a major role to play in the design of a new class of networks and networked systems.

2.1 Generalized Data Independence

A subtle but general issue of timescales underlies data independence. This discussion is not explicit in classical treatments of the topic, but is the key to translating the concept outside of databases proper. At base, data independence is about relative *rates of change* across interfaces, and the software engineering benefits of separating components that change at different rates.

In particular, when data layouts and device configurations change much more frequently than application logic, then it becomes worthwhile to build a layer of intermediating software that automatically adapts to and masks these changes from the applications above. As a mnemonic, we define the following:

[1] When we speak of data independence in this paper, we focus on what is often called *physical* data independence, and which was the focus of Codd's landmark paper [7]. *Logical* data independence refers to the ability to change the logical schema of the database, while preserving the applications' schemata via views. Logical independence was a slightly later development [8]. It is also not quite the "home run" that physical independence is; the impossibility of general updatable views prevents the full realization of logical independence.

Proposition 1 (Data Independence Inequality) *Data independence is beneficial iff*

$$d(\textit{environment})/dt \gg d(\textit{applications})/dt$$

i.e., the rate of change in the computing environment over time (devices, physical data placement and movement, etc.) is much faster than the rate of change in applications that access the data.

The challenge of providing data independence was the genesis of modern database systems research, and has kept the database community productively employed for three decades.

The database community took this challenge to heart because of the typical applications it serves. The need for data independence arises in database settings because the *right-hand* side of the inequality is often so small: many mission-critical database applications (banking, reservations, payroll systems, etc.) operate largely unchanged for *decades*. By contrast, Moore's law assures a steady rate of change in the physical hardware. This is typically compounded by databases being reused over time in unanticipated ways, which often requires modifications to the physical layout (new indexes, partitioning, striping, etc.)

We believe that the inequality above arises again in more general networked settings, for the opposite reason. In many of today's extreme network scenarios, the data independence inequality holds because its *left-hand* side is so large: networks at extreme scales are exceedingly volatile. Moreover, many networking services are naturally data intensive, and these services are growing in complexity – they need more than the simple "level of indirection" provided by traditional OS research. As a result, it would seem that these systems would benefit from the database community's perspective on reusable infrastructures for data independence.

2.2 Pillars of Data Independence

In retrospect, the database systems infrastructure for data independence in database systems was achieved by inventions in two core areas:

- *Indexes* allow value-based lookups to be about as fast as direct lookups, removing the incentive for applications to directly address data. Good indexes *adapt* to changing data distributions, and guarantee efficient, predictable performance without the need for programmer intervention. This kind of index technology was a key enabler of Codd's vision; the B-tree was invented concurrently with Codd's first papers on the relational model, and supplanted non-adaptive schemes like ISAM that gave no formal performance guarantees.
- *Query optimization* allows for efficient execution of declarative languages that support far more than lookup queries ("dereferencing" or "search"). Query optimization is also by nature an adaptive scheme: as data distributions and physical storage parameters change, a query optimizer is able to measure these changes and modify the behavior of applications as a result.

The p2p research community appears to be moving in an analogous direction for massively distributed systems. The first piece seems to be falling into place, through the invention of a class of distributed routing techniques known collectively as *Distributed Hash Tables (DHTs)* (e.g., [29, 26, 32, 27, 19], etc.). DHTs assign values (logical IDs) to machines on a network, and allow packets to be routed "by value" to whichever machine is currently responsible for that value.

The second piece of a data independence agenda remains largely unstudied in the p2p community, and presents a significant opportunity for additional research. However, we should not expect the analogy to be precise: something akin to query optimization may be needed for a large family of networked applications, but there may or may not be an explicit query in sight. We will return to this issue again in Sections 4 and 6.

3 A Quick Introduction to DHTs

DHTs are distributed data structures that provide content-based (a.k.a. "data-centric") routing. The goal is to network a very large, quickly changing set of machines, allowing requests for values (keys) to be correctly routed to a machine currently managing that value. This is to be done without any global knowledge – or permanent assignment – of the mapping of keys to machines. DHTs typically partition a logical key domain among the current set of machines. DHTs typically have the following design constraints:

- *Few Neighbors (degree):* Each node should only maintain a small number of active "neighbors", remembering for each neighbor its physical (IP) address, and its logical "key" partition. A typical constraint for the number of neighbors is $\log n$ for a network of n machines. By keeping this number small, the arrival or departure of nodes from the network results in a bounded number of update messages ($\log n$ for many DHTs).
- *Low Latency (diameter):* Each node should be reachable from any other in a small number of network routing hops; this requires that the neighborhood graph have small diameter. Again, $\log n$ is a typical constraint.
- *Greedy Routing Decisions:* Each node should be able to decide how to forward keyed messages with-

out consulting any other nodes. These greedy decisions must find the "short" ($\log n$) paths between any pair of nodes.

- *Robustness (large min-cut):* As nodes and links come and go, the network should remain mostly connected, and able to route packets. Similarly, it should not exhibit "hot-spots" – overloaded nodes and links. A necessary condition for these features is a network with a non-trivial min-cut, to allow multiple alternate routes for packets.

Most DHTs were invented by cleverly embedding a graph of small degree, small diameter and non-trivial min-cut into some metric space that permits greedy routing decisions. For example, Chord [29] embeds nodes into a unit circle, such that each node's neighbors are at distance $\frac{1}{2^k}$ across the circle for $1 \leq k \leq \log n$. CAN [26] embeds nodes into a d-dimensional torus ($d = \log n$ is typical), so that each node's neighbors are adjacent along different dimensions. Plaxton's algorithm [23, 32, 27] embeds nodes onto the corners of a high-dimensional binary hypercube, so that each node n has one neighbor that is only "far" from n on successively smaller projections of the hypercube: i.e., the value of the ith neighbor of n matches n's value on the first i bits, but may differ otherwise. Variations on these schemes abound; the interested reader is referred to a recent analysis paper [11] for further discussion of these embeddings.

We wish to make a few points here. First, DHTs provide strong theoretical bounds on key "lookup" cost, on routing-table "maintenance" cost, and on robustness to failure. This contrasts with prior routing protocols for p2p overlays, including the widely used "unstructured" p2p networks of Gnutella and KaZaA. While there is work on improving unstructured p2p networks (e.g., [6, 31]), the analogy to B-Trees vs. ISAM seems relevant here: DHTs bring theoretical clarity to a problem previously addressed by heuristics. For the purpose of simple key lookups, it is unclear why one would favor unstructured networks like Gnutella when DHTs can provide strong performance guarantees[2].

A second point is that DHTs are under extreme scrutiny right now, both from algorithm designers and system builders. Proposals for modifications and new designs abound, some of which are important, some less so. We expect there to be significant short-term churn in this area, hopefully followed by a set of "winning" design considerations within a few years. Skeptics are encouraged to periodically revisit this research; many of the initial pragmatic problems with these schemes are being quickly and effectively addressed.

[2]In fairness, unstructured p2p networks were specifically designed for keyword search, not for generic key lookups (Section 5.1.) However, some p2p researchers have proposed unstructured p2p networks as a general-purpose substrate.

4 Evidence: Network Services

Given this introduction to DHTs, we continue by describing a few illustrative core network services that have essentially reinvented simple data independence on the Internet. We show that the data independence inequality is satisfied in these scenarios, and describe how the service "application logic" is decoupled from the physical properties of the changing network – typically via schemes that are analogous to indexes. These examples are intended to be illustrative; there are certainly other such examples in the literature.

4.1 Internet Indirection Infrastructure

The Internet Indirection Infrastructure ($i3$) [28] proposes a generic value-based mapping service that introduces a level of indirection between communicating parties. Receivers use *triggers* to indicate their interest in packets. In their simplest form, $i3$ triggers are pairs (id, r), where id is a globally unique identifier, and r is a node's logical address (key). A trigger (id, r) indicates that all packets with identifier id should be forwarded by the $i3$ infrastructure to the node with address r. This mapping service can, for example, be used to facilitate mobility: as the receiver changes its address from r to r' (e.g. by roaming or by switching devices), the receiver updates its trigger to (id, r'). As a result all future packets with identifier id will be forwarded to the receiver at the new address r'. This way the sender is insulated from the recipient's changes of address. This mapping service can be used for a variety of other networking applications. A 1-to-n mapping of logical addresses to physical addresses supports "anycast": the ability to direct a message to any one of a set of functionally-equivalent servers (e.g. web server replicas). The same mapping can similarly support multicast, where each message is routed to all the addresses mapped to the same id.

At first blush, $i3$ might sound to a database expert like a simple index for providing data independence. A main challenge in this arena is to achieve this "simple" functionality efficiently and reliably at Internet scale and volatility. $i3$ proposes leveraging DHTs to do this. Also, $i3$ has a *service composition* feature that is naturally amenable to query optimization; we will have more to say about this in Section 6.2.

4.2 Intentional Naming and The Like

Naming is considered to be one of the most fundamental and tricky problems in system design, especially for distributed systems [9]. A family of proposals suggest replacing traditional identifier-based naming schemes with declarative schemes ([10, 20], etc.) A well-known recent example of this research thrust is the Intentional

[*sic*] Naming System [1]. This widely-cited idea suggests that a naming system could be built such that applications would access services and devices in an intensional fashion: by description, rather than by name. For example, one might request that a document be sent to "a black-and-white two-sided printer on the 6th floor" rather than to `print626.cs.berkeley.edu:lp0`. This is robust in the face of physical changes to the hardware – if `print626` has failed or was upgraded to color, another printer could be found. Moreover, it is easier for individuals to use, since it unifies service discovery and service invocation. (This latter idea also appears in the "e-services" literature that is currently under investigation in some database circles.) INS supports a semi-structured query language with nested attribute-value pairs; its lookup structure predates DHTs and is significantly more complicated than DHTs. More recent papers from the same group promote the use of DHTs for this purpose [29, 4].

4.3 Content Distribution Networks

Content Distribution Networks like Akamai provide two services: they replicate web pages close to end-users, and provide a data independence service that redirects an HTTP request to a nearby replica – content-based anycast. This data independence is achieved by first hijacking web requests at the DNS level, and then redirecting them to nearby replicas via the *consistent hashing* algorithm [15]. In this environment, the DNS naming scheme offers the "hook" for inserting data independence, and consistent hashing provides the actual indexing scheme that maps URLs to physical objects. Note the clear relationship between routing and indexing in this context: the consistent hashing scheme can be viewed as a distributed index over a 1-to-n mapping, but in effect it is routing network requests to machines with relevant content. Consistent hashing is a precursor to the Chord DHT.

5 More Complex Applications

The previous section showed that a host of core networking facilities are made more robust to variations (in network storage, network connectivity, networked device characteristics, etc.) by providing simple indirections via global-scale indexes. These "Overlay Network" ideas provide network facilities via a layer overlaid on top of IP. In this section we discuss a few examples of more full-service global-scale applications that benefit from more aggressive data independence, and motivate more aggressive machinery (including query optimization) for data independence.

5.1 P2P Filesharing: An Emerging Debate

Filesharing systems need little introduction; they are the most widely-distributed query systems ever built. Filesharing systems provide two basic services: massively distributed keyword search, and point-to-point file transfer. In addition to the much-discussed legal/ethical/business controversy over sharing copyrighted material, the physical design of the keyword search in these systems is also a point of technical debate. Moreover, the popularity of the systems make them a good stress test for large-scale global network applications, and hence a focus of p2p research.

The leading deployed systems (KaZaA, Morpheus, Gnutella, etc.) are based on physical designs in which file names are indexed into a hierarchy in the network: nodes copy their filename list to supernodes ("ultrapeers") in a shallow hierarchy (Gnutella has only 2 levels: peers and ultrapeers). Queries are "flooded" across as many nodes on the root level as is feasible, and these nodes may forward queries on to their children in the hierarchy. One complaint with this architecture is its lack of robustness to failure: if a node high in the hierarchy fails, its children are unreachable until the network readjusts its hierarchy. Another common complaint is that in practice flooding must be limited to a small radius around the query site, sacrificing recall.

Recent proposals suggest using traditional inverted files indexed by DHTs; Overnet is a deployed filesharing system that uses this scheme. Boolean Search proceeds in the traditional way, hashing via the DHT to contact all sites that host a keyword in the query, and doing a distributed join of the postings lists of matching files. This design promises full recall without flooding, but may use significant bandwidth for distributed joins.

There is no consensus as to the best p2p design for these keyword queries. Recent back-of-the-envelope calculations suggest that there is a bandwidth/recall tradeoff between the two, and that neither scheme scales up well enough to do keyword search on a Google-scale workload [16]. This paper also suggests a number of research directions in dynamically optimizing keyword search over DHTs. Of course the data independence afforded by declarative keyword search would admit hybrid unstructured+DHT schemes as well.

5.2 Internet Monitoring

As the Internet has grown, it has become increasingly difficult to understand, measure, protect and control. A variety of projects are attempting to monitor the Internet for both scientific and security purposes ([22, 30], etc.) These projects do modest-scale distributed tracing followed by a degree of centralized data collection, and subsequent application-specific analysis queries. In database terms, these projects are data warehousing efforts, and they face common shortcomings of warehousing: they cannot cover all the possible data sources, and they trade away data freshness for simplicity of query execution. Note that the data volumes in Internet monitoring can make "data-shipping" solutions like warehousing

infeasible without distilling the data significantly.

An alternative is to build a real-time, *distributed* query engine for ad-hoc Internet monitoring queries. This would allow masses of individual end-users to donate some of their cycles and bandwidth for reporting their local streaming view of Internet traffic; these multiple views could be federated in a p2p fashion to support both one-shot analysis queries as well as continuous queries.

This directly maps the traditional data independence problem to networks: declarative network monitoring queries must be optimized to physical query execution schemes. However there is a fundamental new challenge here: the *physical changes occur in the query's intermediate dataflow*, rather than in the base storage. Tuples in an ongoing query plan are being routed all over the Internet, and these routes must be gracefully monitored and adapted for various query optimization metrics.

6 Some Relevant DB Technologies

Modern databases achieved data independence via a mix of techniques that arose over a number of years. The mix has been fairly well codified by now, but a huge investment of invention and experimentation led to today's conventional wisdom. In this section we mention a few of the techniques we are studying – in addition to DHTs – that point the way toward aggressive data independence for networks. These are intended only as illustrative examples; the list neither exhaustively covers current technologies, nor does it purport to provide the union of functionality required for the applications listed above.

6.1 Distributed Aggregation

Data aggregation is an important feature in many network scenarios. It is important for exploratory data analysis, which is a key component of network monitoring, for example. It is also critical for *data reduction*: in many scenarios it is infeasible to ship all the data, but various synopses can be constructed to provide robust approximate answers to queries. Creating synopses on the fly can be done in distributed aggregation frameworks [13].

Standard DHT APIs provide callback interfaces so that application-level code can intercept packets as they are routed. This allows a natural aggregation scheme to be implemented in DHTs. Simply put, every node inserts (sends) their locally aggregated partial state into the DHT, keyed by the identifier of the query site. The DHT routes these partial state tuples in a multi-hop fashion. Before a node forwards a tuple, it intercepts it, combines it with its previously collected partial state, and eventually re-injects collected results back toward the query site. Current DHTs are targeted to point-to-point communication, but is not clear whether their aggregation topologies are efficient and robust.

6.2 Eddies: Commutative Routing

Eddies are an adaptive query processing scheme combining query optimization and execution into a single feedback control loop [3]. In contrast to traditional databases, eddies treat all of query processing as a single adaptable task of routing tuples through (partially) commutative operators. The space of legal query plans is covered by legal eddy routings, but the routing approach allows for simple adaptation at runtime. Moreover, the *SteM* extensions to eddies [24, 18] provide significant additional routing options, corresponding to hybridized join algorithms, competitive access methods, and multi-query optimizations. Because eddies can adapt at fine grain, they can deal gracefully with unknown, volatile physical environments, data distributions, and query mixes.

Although eddies represent a radical departure from traditional database techniques, their routing-based approach is very natural for networks. Moreover, eddies naturally handle the new form of physical volatility mentioned in Section 5.2: they gracefully adapt to physical changes to in-flight query dataflows, a key feature that traditional query optimization does not provide.

As a networking-centric example, consider how eddies complement the service composition model of $i3$. An $i3$ user can inject a packet that specifies a list of services that need to be invoked on some arguments specified in the packet's payload. In $i3$, this list is expressed as a stack of keys, where each key identifies a service. Each routing step is handled by popping the next key off of the stack, and routing to that key.

The $i3$ stack model only supports strict orderings of services, but many services are commutative. For example, a data dissemination system may have to (1) fetch multiple pieces of data from a CDN, and (2) recursively expand email aliases of appropriate recipients. The calls to these services can be interleaved arbitrarily. To accommodate such commutative services, one can consider the specification of partial orders for service composition in $i3$; the partial order for a given packet can be satisfied by one of many possible total orders, which correspond to "query plans", or routes through the network.

The $i3$ routing layer should consider multiple orderings of these routes. It needs to take into account the latency of each routing step, based on the trigger's current location and remaining partially-ordered tasks. Moreover, note that each lookup step may significantly increase the payload – URL lookups expand the trigger with text and embedded images, and the email lookups may expand the trigger with long lists of addresses. Hence the latencies have to be traded off against potential future bandwidth needs – the traditional database notion of "selectivity" maps to payload explosion here.

6.3 Recursive Queries and Network Graphs

Network routing tables are distributed adjacency matrices of logical links. Routing protocols used to maintain these tables (e.g. BGP) are criticized as ad hoc and suboptimal.

There is currently no simple way for nodes to discover multi-hop topology information that could help provide better routing. Some simple route-discovery queries might include "is node X reachable via any IP path?" or "which nodes are k hops away from me in the Gnutella overlay?" (this latter query is especially natural because Gnutella only provides a k-hop query flooding radius). These are recursive queries on graphs. Recursive queries are especially natural in networks, as are materialized views for such queries. As a result, many of the database techniques developed for recursive query processing may be relevant for network routing protocols, particularly those techniques that have been mapped to parallel or distributed scenarios [5].

6.4 Range Search over DHTs

DHTs are analogous to hash indexes, providing only exact-match lookups. An open question is whether range queries can be supported in as graceful and elegant a distributed fashion as DHTs support equality queries. Ratnasamy proposes implementing range queries *over* an arbitrary DHT via a trie-based scheme [25], which hashes data to prefixes of the attributes to be indexed. This is reminiscent of Litwin's Trie Hashing [17], but has an added advantage that the "memory addresses" where buckets of the trie are stored are in fact the DHT keys corresponding to the prefixes. Hence one can "jump" into the trie at any node (key prefix) via a DHT lookup, and do binary search in the key-length to find the appropriate leaf at the start of a range. This provides both efficient lookups, and robustness in the face of failures (missing trie nodes can be "skipped".) In the worst case, this scheme provides range queries with an overhead of $O(\log\log|D|)$ DHT key lookups, where D is the data domain being indexed. More work is required to refine this technique for skewed data sets over large domains. Another simple scheme is to treat DHT keys as addresses of "pages", and directly build a B+-tree over this storage abstraction; some work would be required to make the tree robust to failure (perhaps leveraging a version of Ratnasamy's direct-addressing trick mentioned above.) Alternative schemes have been proposed as well, including a DHT-based range-caching scheme [12], and a technique specifically designed for the CAN DHT based on space-filling curves [2].

7 Infecting the Network, Peer by Peer

Today's most important and widely-used computer systems revolve around the Internet. We have argued here that a core competency of the database community – support for data independence – will play an increasing role in networked system design. We illustrated this point with a number of network services, and proposed a handful of database technologies that may be ripe for exploitation inside networks.

Peer-to-peer systems are a natural "host" for database ideas to infect networking design. The most popular p2p systems are filesharing systems, which are heavily query-driven. Relative to Internet routers, p2p systems are easy to deploy and to influence, and can tolerate the overheads of the fairly detailed data manipulation that is typically done in databases. With some luck, critical ideas may migrate from p2p applications into more ubiquitous parts of the network, perhaps even into the fabric of the network itself.

At Berkeley we're designing a reusable p2p query engine called PIER [14], which is our platform for investigating many of the technical issues and applications mentioned above.

Acknowledgments

I am grateful to various people for (still-evolving) discussions on this theme. Thanks to Ryan Huebsch, Boon Loo, Sylvia Ratnasamy, Timothy Roscoe, Scott Shenker, Ion Stoica, David Culler, Mike Franklin, Hari Balakrishnan, Frans Kaashoek, David Karger, Robert Morris, and the anonymous reviewer.

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proc. 17th ACM SOSP*, Dec. 1999.

[2] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Proc. Second IEEE International Conference on Peer-to-Peer Computing*, Linkoping University, Sweden, Sept. 2002.

[3] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proc. ACM SIGMOD*, pages 261–272, May 2000.

[4] H. Balakrishnan, S. Shenker, and M. Walfish. Semantic-free referencing in linked distributed systems. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2003.

[5] F. Cacace, S. Ceri, and M. A. W. Houtsma. A survey of parallel execution strategies for transitive closure and logic programs. *Distributed and Parallel Databases*, 1(4):337–382, 1993.

[6] Y. Chawathe, S. Ratnasamy, L. Breslau, and N. Lanham. Making gnutella-like p2p systems scalable. In *Proc. of ACM SIGCOMM*, Aug. 2003.

[7] E. F. Codd. A relational model of data for large shared data banks. *CACM*, 13(6):377–387, 1970.

[8] C. J. Date and P. Hopewell. File definition and logical data independence. In *Proc. ACM SIGFIDET*, pages 117–138. ACM, 1971.

[9] M. J. Flynn, J. Gray, A. K. Jones, K. Lagally, H. Opderbeck, G. J. Popek, B. Randell, J. H. Saltzer, and H.-R. Wiehle, editors. *Operating Systems, An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*. Springer, 1978.

[10] D. K. Gifford, P. Jouvelot, M. Sheldon, and J. O'Toole. Semantic file systems. In *Proc. 13th ACM SOSP*, Oct. 1991.

[11] K. Gummadi, R. Gummadi, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *ACM SIGCOMM*, Aug. 2003.

[12] A. Gupta, D. Agrawal, and A. E. Abbadi. Approximate range selection queries in peer-to-peer systems. In *Proc. First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, CA, Jan. 2003.

[13] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Toward sophisticated sensing with queries. In *Proc. Information Processing in Sensor Networks (IPSN)*, 2003.

[14] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with PIER. In *Proc. 29th International Conference on Very Large Data Bases (VLDB)*, 2003.

[15] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*, 1997.

[16] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[17] W. Litwin. Trie hashing. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 19–29, Ann Arbor, Michigan, Apr. 1981.

[18] S. R. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proc. ACM SIGMOD*, 2002.

[19] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Mar. 2002.

[20] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus–an architecture for extensible distributed systems. In *Proc. 14th ACM SOSP*, pages 58–68, Dec. 1993.

[21] C. H. Papadimitriou. Database metatheory: Asking the big queries. In *Proc. 14th Symposium on Principles of Database Systems (PODS)*, pages 1–10, May 1995.

[22] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An architecture for large-scale internet measurement. *IEEE Communications*, 36(8):48–54, Aug. 1998.

[23] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1997.

[24] V. Raman, A. Deshpande, and J. M. Hellerstein. Using state modules for adaptive query processing. In *Proc. IEEE International Conference on Data Engineering (ICDE)*, 2003.

[25] S. Ratnasamay, J. M. Hellerstein, and S. Shenker. Range queries in DHTs. Technical Report IRB-TR-03-009, Intel Research, July 2003.

[26] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. 2001 ACM SIGCOM Conference*, August 2001.

[27] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[28] I. Stoica, D. Adkins, S. Zhaung, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. ACM SIGCOMM*, Aug. 2002.

[29] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: Scalable Peer-To-Peer lookup service for internet applications. In *Proc. 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[30] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, Oct. 1999.

[31] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proc. of the 19th International Conference on Data Engineering (ICDE)*, Mar. 2003.

[32] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001.