Musings Upon the Theme of Peer-to-Peer Search

Elth Ogston Konrad Iwanicki Maarten van Steen

Department of Computer Science
Vrije Universiteit Amsterdam,
De Boelelaan 1081A, 1081 HV Amsterdam, The Netherlands
{elth, iwanicki, steen}@few.vu.nl

Keywords: peer-to-peer, search algorithms, decentralization

Abstract

Efficient search is an active topic in peer-to-peer networks. To date most of the work in this area has been oriented towards devising new algorithms. However, little has been done to understand the fundamental similarities and differences between the wide variety of proposed designs. Such an understanding would be invaluable for the architects of future decentralized systems. In order to stimulate further research in this direction, we present an overview of six example systems whose designs cover the majority of current research themes in peer-to-peer search. The diversity of the presented approaches demonstrates the breadth of the design space available. Based on these examples, we identify a number of open issues that require deeper exploration.

1 Introduction

Searching is an important function in any information system, but the task of creating efficient search algorithms is a stumbling block for many systems aiming at decentralization. Peer-to-peer (P2P) networks are an important example. They attempt to achieve decentralization in a setting that emphasizes scalability, robustness, self-organization, and autonomy. These design goals make search in P2P networks an especially challenging problem, as the behavior of nodes participating in the network can only be predicted and controlled to a very limited extent. In response to this challenge, numerous P2P search algorithms have been devised. The most adopted classification of these algorithms divides them into *structured* and *unstructured* approaches [1, 4], depending on the data distribution pattern imposed and the resulting network topology.

In our view, the design goals that search algorithms address and the design choices these goals inform are fundamental characteristics on which classification should be based. Yet, the structured-unstructured distinction fails to describe these important aspects. This classification method is therefore of limited practical use to P2P system designers. It does not explain the role of data-query models in supporting fully decentralized search. Nor does it consider the differences between moving data to the sources of queries and moving queries to data, or between manipulating network topology and changing data distribution. The structured-unstructured distinction also does not address issues associated with replication, such as consistency. We suspect that these are just a few examples of the significant practical differences among algorithms that should be captured in a study of P2P search. For this reason we are currently surveying existing P2P search algorithms from the perspective of the design choices they make, rather than the resulting network topologies and data distributions.

In this paper we attempt to motivate the need for such a survey by presenting some examples of the open questions that we have encountered to date (Section 8). In order to shed some light on the origin of such questions, we present preliminary analyses of six representative algorithms for implementing search in P2P systems.

- Pastry [5] is an Internet-wide distributed hash table that executes queries with low latency and high recall (Section 2).
- Harren et al. [3] discuss implementing relational queries (similar to SQL in databases) on a distributed hash table (Section 3).
- Cooper and Garcia-Molina [2] explore the possibility of reducing the total load of a Gnutella node while providing load balancing, and preserving Gnutella's original properties of minimal

network maintenance cost and complexity (Section 4).

- Sripanidkulchai et al. [6] present a version of Gnutella that reduces network load and query execution latency (Section 5).
- Astrolabe [7] is a system for supporting aggregation queries (Section 6).
- Newscast [8] provides simple and robust information dissemination (Section 7).

For each algorithm, we investigate the assumptions behind the design goals, discuss the resulting design choices, and enumerate the difficulties that arise from those choices. There are apparent trade-offs between various aspects, for instance, between query execution latency and recall, between system maintenance cost and the applicability of an algorithm to different types of data, and between fairness and autonomy. The variety found among just six examples illustrates the breadth of possibilities for designing a P2P search system and underscores the need for an in-depth exploration of the topic.

2 Pastry

Pastry [5] has been chosen for the purpose of the paper as an example of a distributed hash table (DHT). One common use of a DHT is implementing a file storage system (in case of Pastry, the associated project is called PAST) in a P2P setting. As in a centralized case, files are named with a unique identifier, by which they can be located. To preserve distribution transparency, the system should guarantee that a file is found if it exists, and moreover, this location process should be fast. In the terminology of searching, the system should provide high query recall and low latency. In order to utilize the available storage space, as provided by all nodes, in an efficient way, a further goal is to distribute files evenly.

To provide high recall, PAST organizes the storage according to file identifiers, such that each file has a well-defined set of nodes on which it can be located. Additionally, by ensuring a uniform distribution of file identifiers, the storage utilization is balanced between nodes. More specifically, when joining the system, each node is assigned a large, fixed-size, uniformly random number. Similarly, each file is named with a large number. This name is a value of a cryptographic hash function, like SHA-1, over the contents of the file. This hash function creates a uniform distribution of file identifiers. When a file is inserted to the system by a node, it is forwarded (routed) to a node with a number that is numerically closest to the identifier of this file.[1] This node stores the file. The same routing algorithm is used for forwarding queries. This deterministic method of placing and locating files guarantees that files that are in the system will always be found.

By organizing the network, such that each node knows a short and low-latency path to any other node in the network, PAST guarantees fast responses to queries. Because the network can be very large, a node cannot store information about every other node. Instead, it establishes links with a subset of nodes, referred to as neighbors. PAST is very restrictive about which neighbors a node must choose. More specifically, a node's number can be represented as a string consisting of hexadecimal digits. For each prefix of length l of this string, the node establishes links with other nodes whose numbers share the same prefix. One neighbor is chosen for each possible value of the $l+1$-st digit. Each node routes a query for a file with a given identifier by resolving subsequent prefixes of the identifier, that is, it chooses a neighbor with the longest prefix that matches the identifier. Since each time a query is forwarded, at least one digit is resolved, a query can be forwarded at most $O(log_{16}N)$ times, where N is the total number of nodes in the network. Additionally, by selecting neighbors with low-latency connections, the latency of the path the query must follow is minimized.

In essence, the design of PAST is founded on a fixed, structured data-query model. The simplicity and structure of the data-query model allows a straightforward, predefined network organization to be built. Enforcing this organization requires that data and queries be moved to specific nodes. Main advantages of this design are low routing latency, high query recall, and balanced storage utilization. However, a user may query for a data item only by specifying its identifier. Such a simple data-query model can be a major disadvantage as it limits applicability. In addition low query execution cost is achieved at the expense of complex operations necessary to maintain the network organization upon changes in node membership and when new data are added.

3 Complex Queries on DHTs

Harren et al. [3] make a number of proposals that address one of the aforementioned disadvantages of DHTs', namely their extremely simple data-query model. The ability to locate an object given its name is a severely limited form of search, especially if the name must be represented as a unique number. More complete data-query models allow objects to be located according to a variety of attributes. They also include operations in which a reply requires analyzing multiple objects, for instance, a query for an object with a maximal value of some attribute. Harren et al., thus, choose to study the possibilities for support-

[1] For the reasons of fault tolerance, PAST actually stores the file on multiple such nodes.

ing a broad range of queries on a DHT with minimal extensions to the standard functionality.

The object naming system used in DHTs is not user-friendly. Looking up a file requires knowing the content-based unique number under which the file has been stored. Consider for example a file sharing system. Users typically would like to search for a file according to a human-readable file name. They would also like to be able to find a file for which they only know an approximate name. Harren et al. discuss a technique called "N-gram Indexes" that addresses these issues. This technique can be used to implement partial match substring searches on a DHT. Files are given text names which are split into all possible substrings of a length N. For N = 4, for instance, the text name "Madonna - Hung Up" is split into 14 strings: "Mado", "adon", "donn", "onna", "nna ", "na -", "a - ", " - H", "- Hu", " Hun", "Hung", "ung ", "ng U", and "g Up". The file is then associated with, and can be found using, each of these substrings. By issuing a query for all 14 substrings, the user will still receive the desired file, even if the the query text does not exactly match "Madonna - Hung Up". However, since each substring is probably associated with multiple files, many unwanted files may be returned, requiring additional selection. The presented scheme essentially demonstrates how a file stored in a DHT can be searched for using multiple attributes.

In order to implement more complex query functions, Harren et al. suggest adding an iterator to the standard DHT interface. Such an iterator scans through all DHT entries on a given node. By executing scans in parallel on all machines, all of the files can be examined. It enables implementing operations known from databases, for example, selection, joins, grouping and aggregation.

This work suggests that it is possible to implement a complex data-query model on a DHT, but at a fairly high price. In order to support multiple attributes, multiple directories, one per attribute, are created. This increases storage consumption, maintenance cost and query execution cost. The scan operation enabled by the iterator function has similar disadvantages, as it has to gather information from possibly all nodes.

Essentially, these problems highlight the trade-offs between search generality and decentralization. By allowing for more attributes the original simplicity of locating a particular file in the DHT is lost. Additionally, the majority of complex query operators require an overview of all data objects. Thus, by attempting to mimic centralized search, which often requires global knowledge, certain advantages of decentralization are lost. In particular, adding more resources/nodes to the system does not necessarily increase the processing and storage capacity of the system.

4 Ad Hoc, Self-Supervising P2P Search Networks

The Gnutella file sharing system is the archetypical example of an "unstructured" P2P network. No restrictions are placed on on the location of files within a Gnutella network, or on the links that may be established between nodes. Instead, nodes store files in which they are interested, and are connected to a number of other arbitrarily chosen nodes. Search is done by flooding queries through the network. The flooding algorithm requires a node to send a query to all of its neighboring nodes. These nodes also forward the query to their neighbors, and so on. In order to prevent queries from possibly circling ad infinitum a maximum hop count is specified to limit the number of times a query can be forwarded. No guarantees are made about search recall. Instead, the developers of Gnutella simply assumed that the most popular files are widely replicated, and therefore most queries quickly return results.

Flooding search is very expensive in terms of the query traffic it generates, and thus the scalability of Gnutella has always been questioned. Cooper and Garcia-Molina [2] argue that, despite this drawback, Gnutella is an attractive system because of its simplicity, flexibility, and robustness. They reason that flooding costs can be reduced by optimizing the network topology. In the real world not all nodes constituting the network have similar capabilities, therefore naturally, the more powerful ones can take over some work from the less powerful. This fact has also been previously applied through the idea of super-peers, powerful nodes that gather all metadata necessary for searching from all nodes connected to them, that is, their clients. Super-peers then process all requests issued by their clients based on the gathered information (so-called search metadata) or by flooding queries among themselves. However, in practice it may be difficult to decide whether a node should be a super-peer, and if so, how many clients it should serve. Therefore, Cooper and Garcia-Molina propose a solution in which nodes self-tune the topology of the network by making purely local and selfish decisions. They show that this improves load distribution and reduces the total load on the system.

Load redistribution according to node capabilities is achieved by rearranging links. A node can establish a link to any other node in the network using a `connect()` operation. If a node becomes overloaded because it has too many neighbors, it uses a `break()` operation to drop some links. The disconnected node may then re-`connect()` to other nodes until it determines it has enough links. The combination of `connect()`s to random nodes and deliberate `break()`s, makes the network self-tuning, since the overloaded nodes shed load, while the other ones pick

up the slack.

Load reduction is achieved by replicating search metadata, thus effectively moving data towards probable sources of queries. When sending search metadata to their neighbors, nodes shed load by asking their neighbors to take on some of the query processing. Similarly to queries, metadata is communicated through links. For a node to search other nodes, the node either needs to send queries to these nodes, or receive their metadata, enabling local search. Only when two nodes want to search each other, do the query and metadata links need to be the same. Therefore, a distinction between metadata and query links is made. This approach allows search metadata to be created and propagated dynamically in a way that best improves the efficiency of the network.

Cooper and Garcia-Molina demonstrated that the described algorithm for rearranging links and moving metadata effectively redistributes and reduces load compared to the original Gnutella. It also improves on the previous super-peer solutions. It should be emphasized that this system would be considered unstructured according to the common classification, whereas in fact, the main idea is to create a dynamic structure relating to the underlying resources and their usage.

5 Content Location Using Interest-Based Locality

Sripanidkulchai et al. [6] introduce another approach to adding structure to the basic Gnutella network. Section 4 describes an algorithm that adapts network topology to the structure of the underlying physical network. Sripanidkulchai et al. suggest that many applications exhibit inherent "structures" in the way data is accessed. In particular, they posit the existence of "interest-based locality" in file-sharing applications. Interest-based locality refers to the phenomena that particular files tend to be collocated, that is, two users often happen to store the same files because they share similar interests. Thus if a user has previously found interesting content on a node, there is a good chance that the same node contains files matching subsequent queries.

Accordingly, Sripanidkulchai et al. propose adding "interest-based shortcuts" between nodes. The basic idea is that, in addition to the default arbitrary links, a node maintains a shortcut list of neighbors that have previously replied to its queries. Shortcuts are ranked according to their usefulness, measured as the ratio of the number of replies provided to the number of queries received. When a new query is issued, it is first sent to these shortcut-neighbors in a sequential manner, in order of their ranking. If none can answer the query, the default flooding mechanism is resorted to.

Sripanidkulchai et al. evaluate their design on a series of traces from web browsing and P2P file sharing applications. The use of interest-based shortcuts reduces overall query traffic because, for many queries, shortcut links eliminate the need for flooding. It is not entirely clear, however, if this improvement is due to interest-based locality in the data sets studied, or to some other characteristics of user behavior. Firstly, web browsing traces, for which the best results were achieved, are likely to exhibit temporal locality; a series of HTTP get requests is related to a user's current task, but not necessarily to his long term interests. Simulations with these traces show that the average hop count necessary to get an answer is only slightly greater than one, when using shortcuts. This means that in most cases the first shortcut link chosen gives a desired result. Therefore, simply sending each query to the node that returned the last reply may provide about the same benefits as interest-based shortcuts. Secondly, file-sharing data sets tend to exhibit a very uneven distribution of files: a small number of nodes store a large number of files, while the majority of nodes store very few files. In this case the shortcut scheme would result in all nodes linking to those few that hold most of the system content, independently of users' interests.

In general, patterns in user behavior are an interesting source of structure for P2P systems. However, decentralized heuristics for discovering these patterns are difficult to devise. The patterns to which an algorithm adapts the network topology may not exist in practice. Even when they do occur, the cost of topology adaptation may outweigh the possible benefits. Worse yet, other unpredicted aspects of user behavior can cause a system to adapt to a counter productive configuration. Such a situation can be observed, for example, when simulating the interest-based shortcut algorithm on the file-sharing traces. The resulting topology, in which all nodes are linked to the few nodes that store large numbers of files, starts to resemble a centralized configuration.

6 Astrolabe

In all search algorithms presented so far, queries are routed from node to node until the relevant data is encountered. In contrast, Astrolabe [7] is an example of a system in which data is moved to users in anticipation of future queries. Astrolabe has been designed to monitor distributed state; more specifically, it tracks and aggregates changes in the state of resources bound to particular machines. In the case where the rate of such changes is unknown or may vary, pushing aggregate data towards interested users can be more efficient than periodical polling.

Resources maintained by nodes are organized into groups, referred to as zones. In order to have different

levels of aggregation, these zones form a tree structure. Every node is a leaf in this tree and has a unique name describing its "position" in the tree, and thus, its membership in particular zones. For example, a node named "/nl/amsterdam/vu/note-konrad" constitutes a leaf zone "/nl/amsterdam/vu/note-konrad", and is also a member of four internal zones: "/nl/amsterdam/vu/", "/nl/amsterdam/", "/nl/", and "/". This last zone is also called the root zone.

The goal of Astrolabe is to provide users with flexible, customizable aggregate information about the state of interesting resources in particular zones. Aggregation is a key concept, as it allows the amount of information exchanged in the network to be bounded. For example, a user may ask to count the nodes whose resources satisfy certain conditions, or even select some random three such nodes, but he cannot ask to list all of them, since that list could be of unbounded size.

Each zone has a set of attributes that can be queried by the users of the system. Attributes of a leaf zone represent the state of some resources of the node corresponding to this zone. Attributes of internal zones represent aggregated values of the appropriate attributes of the child zones. Attribute values of a leaf zone are updated directly by the corresponding node, whenever the resource state of this node changes. Internal zones, however, are not associated with a specific node. The information about an internal zone is therefore replicated among a well-specified set of nodes, such that zones higher in the zone tree have more replicas.[2] Updates of the attribute values of an internal zone are performed by a dynamically elected representative node belonging to this zone.

To propagate changes of attribute values an epidemic protocol is employed. Each node periodically chooses some other random node to gossip with, that is, to exchange some information with. These two nodes store, among other things, attribute values of all zones up to and including their least common ancestor. However, the freshness of these values can vary. Gossiping between two nodes allows them to each adopt the most recent attribute values. Repeated gossiping quickly spreads information throughout the system, so any later query for an attribute in a zone may be served locally by any member of this zone.

The set of queries the system uses for computing aggregates is not fixed. Any user can program a new query in an extended variant of SQL. Query code is then embedded in so-called aggregation function certificates (AFCs), which are installed in appropriate zones. A user interested in a particular state of a particular resource in a particular zone creates an AFC that is installed in this zone. This operation can introduce new attributes which may be either results of aggregate functions on existing attributes, or new functions on some resources, possibly newly added to the system. AFCs are propagated throughout the child zones by the same gossiping protocol as used for attribute value propagation. Thereby, all interested nodes dynamically adjust their functionality. If the user later wants to obtain information about the resource of interest, he checks locally the value of the attributes introduced by the AFC.

Astrolabe has two main advantages: a flexible data-query model and fast query execution. First, Astrolabe is able to handle aggregation, one of the most complex query functions. AFC installation enables the addition of new functionality and attributes at runtime. The resulting flexibility allows the system to be utilized not only for aggregation, but also, for example, for leader election, voting, multicast routing, resource location, and object placement. Second, installing queries enables the system to move data proactively to nodes that may need it, which results in fast query execution. However, the amount of replication necessary and the use of a simple epidemic protocol for updating replicas constrains the size of attribute values and requires a significant sacrifice of data consistency. Consequently, Astrolabe only supports eventual consistency, which does not guarantee the freshness of data. Moreover, the aggregated values may be computed from information obtained at different times. Finally, the zone tree, which bounds data propagation thus limiting network traffic, must be designed and reconfigured by a system administrator.

7 Newscast

Information dissemination addresses the problem of moving data to the location where it is required. Astrolabe (described in Section 6) moves data in an orderly manner, whereas Newscast [8] investigates the concept of unstructured data dissemination. In Newscast each node maintains a set of data items. These data items are spread by periodical gossiping, that is, data exchanges between a node and its neighbors. The goal, as we explain later, is to have data items from every node reasonably fairly disseminated throughout the network.

The idea behind Newscast can be thought of as a concept dual to Gnutella (see Section 4), in which queries are the things disseminated. Gnutella assumes that there are multiple randomly located sources of most data items, and thus a query moving through the network will quickly find a copy of the data item it requires. Conversely, Newscast disseminates data, on the assumption that there are multiple randomly

[2] Each node stores attribute values of all zones on the path from the corresponding leaf zone to the root zone, as well as the attribute values of every sibling zone of these zones. For example, every node has a local copy of the attribute values of the root zone and all child zones of the root zone.

located sources of most queries, and thus data moving through the network will quickly reach locations where it is needed.

The Newscast protocol works as follows. Each node maintains a fixed-sized set of c data items gossiped periodically at a fixed rate, ΔT. During gossiping a node (1) adds one new data item to its cache, stamped with its current time; (2) sends all data items to some neighbor; (3) receives all of that neighbor's items; (4) rejects some items so that it keeps only the c most recent ones (as determined by the time-stamps). The node's neighbor in the exchange also keeps the same set of items. The neighbor to gossip with, as chosen in point (2), is picked at random among a set of nodes the node knows about. This set is determined by the data items the node stores. More specifically, the data items nodes publish are labeled with the address of their publisher. The data item set of a node is thus also the neighbor set of that node. Consequently, when data items are moved between nodes, the links between nodes also change.

Information dissemination can be a building block for membership management protocols. Membership information is simply another type of data to be propagated throughout the system. This is in fact done in the Newscast protocol when changing links through data movement. Adding a data item published by a new node, automatically adds this node to the system. Nodes that stop publishing are also automatically removed as their data becomes stale and is evicted.

The goal of Newscast is to disseminate information such that each node has an equal chance of seeing some data item from every other node. This implies that over a sufficiently large period of time a node is likely to see data items produced by all other nodes. To accomplish this, Newscast tries to ensure two properties: that an equal number of replicas of data items published by each node exist, and that these items are uniformly distributed across all nodes.

By having each node publish and exhange data items at the same rate, and by means of the time-based eviction policy for the data items, Newscast aims at establishing the first property. Note that this mechanism is independent of the way the items are propagated. In particular, employing it in a network based on a static topology still evenly divides storage space among items produced by different nodes.

Achieving the second property is more difficult. Flooding with a limited hop count, as implemented in Gnutella, often bounds the set of nodes that can see information published by a particular node. Aggressively changing links, as done in Newscast, ensures that data is routed to a continuously changing set of nodes. That is, the limits on the lifetime of a data item, specified either by a hop count or an item eviction policy, do not necessarily bound the potential locations of that item.

The biggest advanage of Newscast's membership management mechanisms is robustness to node failure. This robustness comes from the property that each node always has a neighbor list that represents an approximately random sample of all network nodes. Moreover, similarly to Gnutella, Newscast is a system in which the participation requirements for nodes are minimized. However, also as in Gnutella, this advantage comes at the price of high communications costs of information dissemination, and a lack of guarantees about when, if at all, data will reach a particular location.

8 Challenges

As the above discussion shows, search in P2P networks is a complex problem. There is a wide variety of possible choices that the designer of a search system can make. This design space cannot be fully captured by simply dividing systems according to the structured-unstructured classification. For example, this classification does not address the following questions:

1. What are the possible principles upon which network topology and data distribution can be based?

2. How can migrating data instead of routing queries affects attainable objectives?

3. How do data-query models affect possible ways of achieving decentralization?

All but a basic brute force approach (as in the original Gnutella) require some form of structure. Thus, in order to answer the first question, we need a broad understanding of the advantages and the disadvantages of various organizations. At the minimum, possible effective sources of the system structure must be identified. For example, interest-based shortcuts, Section 5, assume that the data distribution follows an inherent semantic pattern. On the other hand, the dynamic topology optimizations discussed in Section 4 employ the capabilities of nodes as the source of system organization. Conventionally, these are both unstructured designs. Pastry (Section 2) and Astrolabe (Section 6), both conventionally structured designs, again use different criteria as the basis for their organization, namely, the data-query model, in the case of Pastry, and a domain-related hierarchy, in the case of Astrolabe.

Another facet of the design space concerns what is being moved by the system in order to match queries with data. DHTs (Section 2, 3) route both queries issued by the user and newly inserted data. The described Gnutella-based systems (Section 4, 5) only route queries, but both change links as a result of those queries. Astrolabe (Section 6) and Newscast

(Section 7) primarily move data, but Astrolabe also disseminates changes to the data-query model in the form of new AFCs, whereas Newscast changes links. These differences originate from the differences in the objectives of the systems. For example, DHTs and Astrolabe aim at providing guarantees on recall, while Gnutella-based approaches and Newscast are more concerned with flexibility.

It is also clear that the "complexity" of the data-query model that the system provides has a large impact on design. This can be mainly observed when contrasting the a plain DHT (Section 2) with SQL-based queries executed on the same network topologies (Section 3) and with Gnutella (Section 4).

The above discussion shows that the above three questions are still open. Furthermore, these are certainly not all of the aspects that need consideration when developing a decentralized search system. Our current research activities focus on creating a detailed characterization of the design space. The ultimate goal is to be able to compare systems that at first glance do not exhibit any similarities, and contrast systems which appear to work in a similar fashion. This detailed understanding of the purposes and consequences of various design choices could enable better future solutions.

References

[1] ANDROUTSELLIS-THEOTOKIS, S., AND SPINELLIS, D. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR) 36* (December 2004), 335–371.

[2] COOPER, B. F., AND GARCIA-MOLINA, H. Ad hoc, self-supervising peer-to-peer search networks. *ACM Transactions on Information Systems (TOIS) 23* (April 2005), 169–200.

[3] HARREN, M., HELLERSTEIN, J. M., HUEBSCH, R., LOO, B. T., SHENKER, S., AND STOICA, I. Complex queries in DHT-based peer-to-peer networks. In *Proceedings of the First Int. Workshop on Peer-to-Peer Systems (IPTPS)* (Cambridge, MA, USA, March 2002), pp. 242–259.

[4] RISSON, J., AND MOORS, T. Survey of research towards robust peer-to-peer networks: Search methods. Tech. Rep. UNSW-EE-P2P-1-1, Dept. of Electrical Engineering, Univ. of New South Wales, Sydney, Australia, September 2004.

[5] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)* (Heidelberg, Germany, November 2001), pp. 329–350.

[6] SRIPANIDKULCHAI, K., MAGGS, B., AND ZHANG, H. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceedings of the 22nd IEEE INFOCOM Conf.* (2003), IEEE Computer Society Press.

[7] VAN RENESSE, R., BIRMAN, K. P., AND VOGELS, W. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems (TOCS) 21*, 2 (May 2003), 164–206.

[8] VOULGARIS, S., JELASITY, M., AND VAN STEEN, M. A robust and scalable peer-to-peer gossiping protocol. In *Proceedings of the Second Int. Workshop on Agents and Peer-to-Peer Computing (AP2PC)* (Melbourne, Australia, July 2003), pp. 47–58.