Improving Data Access in P2P Systems

The P-Grid approach enables distributed search and replication. Gridella, a P2P system based on P-Grid, improves on Gnutella's search performance while reducing bandwidth requirements.

Karl Aberer and Magdalena Punceva
Swiss Federal Institute of Technology

Manfred Hauswirth and Roman Schmidt
Technical University of Vienna

The limitations of client-server-based systems become evident in an Internet-scale distributed environment. Resources are concentrated on a small number of nodes, which must apply sophisticated load-balancing and fault-tolerance algorithms to provide continuous and reliable access. Additionally, network bandwidth must be increased steadily to handle requests to and from successful Internet servers. Caching and replication were introduced a posteriori to remedy these problems in a client-server setting when the World Wide Web, as the most successful Internet service, developed into a network bandwidth nightmare.

Peer-to-peer systems offer an alternative to traditional client-server systems for some application domains. In P2P systems, every node (peer) of the system acts as both client and server (servent) and provides part of the overall information available from the system. The P2P approach circumvents many problems of client-server systems but results in considerably more complex searching, node organization, security, and so on. Napster, which made the P2P idea popular, avoids some of this complexity by employing a centralized database with references to files on peers. Gnutella, another well-known P2P system, has no central database, but requires a communication-intensive search mechanism. (See the sidebar, "The Gnutella File-Sharing System," page 60, for a discussion of the Gnutella protocol.)

In this article we present Gridella, our Gnutella-compatible P2P system. Gridella is based on the Peer-Grid (P-Grid) approach, which draws on research in distributed and cooperative information systems to provide a decentralized, scalable data access structure.[1] Gridella improves the highly chaotic and inefficient Gnutella infrastructure with directed search and advanced concepts, thus enhancing efficiency and providing a model for further analysis and research.

P-Grid: Distributed Search and Replication

P-Grid is a virtual binary search tree that distributes replication over a community of peers and supports efficient search –

 1089-7801/02/$17.00 ©2002 IEEE

that is, search time and number of generated messages grow as $O(\log_2 n)$ with the number of data items n in the network.[1] Unlike peers in other approaches that construct scalable, tree-based, distributed indexing structures,[2-4] peers in P-Grid perform construction and search/update operations without any central control or global knowledge in an unreliable environment.

Distributed Search Structure

P-Grid's search structure exhibits the following properties:

- it is completely decentralized;
- all peers serve as entry points for search;
- interactions are strictly local;
- it uses randomized algorithms for access and search;
- probabilistic estimates of search request success can be given;
- search is robust against node failures; and
- it scales gracefully in the total number of nodes and data items.

In P-Grid, each peer holds only part of the overall tree, which comes into existence only through the cooperation of individual peers. Every participating peer's position is determined by its path, that is, the binary bit string representing the subset of the tree's overall information that the peer is responsible for. For example, the path of Peer 4 in Figure 1 is 10, so it stores all data items whose key begins with 10.

The paths implicitly partition the search space and define the structure of the virtual binary search tree. As Figure 1 illustrates, multiple peers can be responsible for the same path. Peer 1 and Peer 6, for example, both store keys beginning with 00. Such replication improves the P-Grid's robustness and responsiveness because we assume that peers are not always online, but rather with a certain, possibly low, probability.

P-Grid's routing approach is simple but efficient: For each bit in its path, a peer stores the address of at least one other peer that is responsible for the other side of the binary tree at that level. Thus, if a peer receives a binary query string it cannot satisfy, it must forward the query to a peer that is "closer" to the result. In the example P-Grid, Peer 1 forwards queries starting with 1 to Peer 3, which is in Peer 1's routing table and whose path starts with 1. Peer 3 can either satisfy the query or forward it to another peer, depending on the next bits of the query. If Peer 1 gets a query starting with 0, and the next bit of the query is also 0, it is responsible for the query. If the next bit is 1, however, Peer 1 will check its routing table and forward the query to Peer 2, whose path starts with 01.

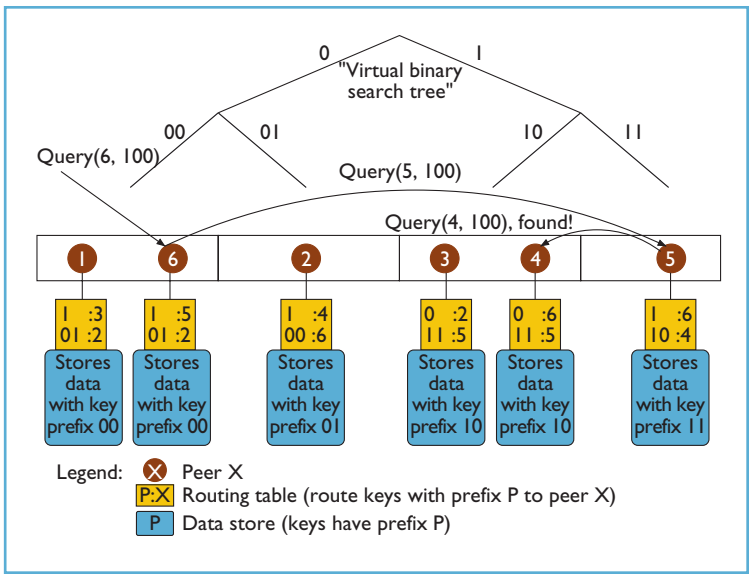

Figure 1. Example P-Grid. Each peer is responsible for part of the overall tree. When a peer receives a query it cannot answer, it refers to its routing table to find the appropriate peer to forward the request to.

The P-Grid construction algorithm (described later) guarantees that peer routing tables always provide at least one path from any peer receiving a request to one of the peers holding a replica so that any query can be satisfied regardless of the peer queried. Figure 2 illustrates this property for the P-Grid in Figure 1. In the P-Grid network shown, no path between Peer 3 and Peer 1 exists, but there is a path from Peer 3 to Peer 6, which holds the same data as Peer 1.

Search requests in P-Grid are sent to arbitrary peers. In Figure 1, a query for 100 is sent to Peer 6. Because Peer 6 is responsible for keys starting with 00, it checks its routing table for the longest common prefix, which is 1, and forwards the query to Peer 5. In a real setup, multiple peers would be listed for each prefix in the routing table, and the peer receiving the query would forward it to a peer randomly selected from this set. Without constraining general applicability, we assume in this simple example that each prefix is serviced by one peer entry in the routing table. Upon receiving the query, Peer 5 does the same checks as Peer 6 and forwards the query to Peer 4, which has the longest common prefix in its routing table. Because Peer 4 has no longer common prefix in its routing table, it searches its local data store for data with the key 100. If the key exists, Peer 4 returns a reference to the associated data to the original requester, Peer

The Gnutella File-Sharing System

Gnutella is a decentralized file-sharing system whose participants form a virtual network and communicate peer-to-peer via the Gnutella protocol (http://www.clip2.com/GnutellaProtocol04.pdf) for distributed file search.

To participate, a peer first connects to a known Gnutella host. Gnutella has five basic message types:

- *Ping* — announcement of availability and probe for other servents.
- *Pong* — response to a ping.
- *Query* — search request.
- *QueryHit* — returned by servents that have the requested file.
- *Push* — file download requests for servents behind firewalls.

Upon receiving a message, the servent decrements the message's time-to-live field. If the TTL is greater than 0 and the servent has never seen the message's identifier (loop detection), it resends the message to all known peers. The servent also checks whether it should respond to the message. If it receives a *Query*, for example, it checks its local file store and responds with a *QueryHit* if it can satisfy the request. Responses are routed along the same path as the originating message.

In a simplified Gnutella session, servent *A* connects to servent *B* and sends a *Ping* message. *B* responds with a *Pong* and forwards the *Ping* to its peers *C* and *D*, who respond with another *Pong*. After some time, *A* knows a number of servents and vice versa. It routes messages as described above and may initiate queries. When it receives a *QueryHit*, it tries to connect directly to the specified servent and runs a simplified HTTP GET interaction to retrieve the file. If the requested servent is behind a firewall, *A* might send a *Push* message (along the path of the *QueryHit*). The *Push* message specifies where the firewalled servent can contact the requesting servent to run a passive GET session. If both servents are behind firewalls, the download is impossible.

Technical Problems

From a user's perspective, Gnutella is simple and effective because hit rates for search queries are reasonably high, it is fault tolerant toward servent failures, and it adapts well to dynamically changing peer populations. From a networking perspective, however, the price is very high bandwidth consumption. Search requests are broadcast over the network and each node receiving a search request scans its local database for possible hits.

Assuming a typical TTL of 7 and an average of four connections *C* per peer (that is, each peer forwards messages to three others), the total number of messages originating from one Gnutella message (including the responses) can be calculated as:

continued on p. 61

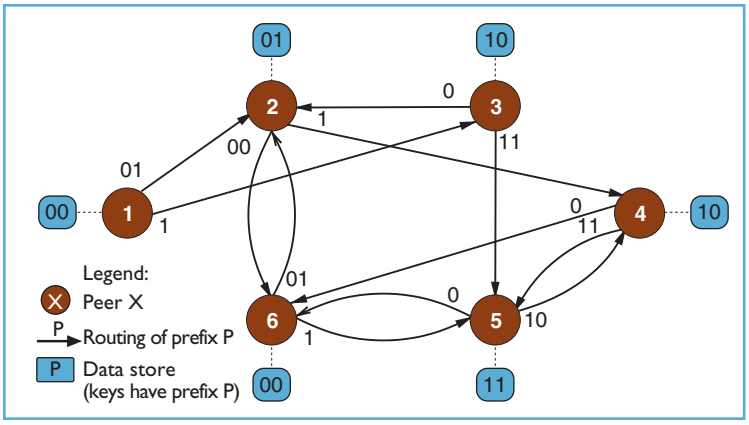

Figure 2. Example P-Grid network. Peer routing tables provide at least one path from any peer receiving a request to one of the peers holding a replica so that any query can be satisfied regardless of the peer queried.

6, which can then request the data. Thus, the search order is equivalent to a binary tree search regardless of the query's entry point.

Search Algorithm

Figure 3 shows P-Grid's search algorithm. The parameter `peer` indicates the address of the peer to send the query to, `query` is the search string, and `index` indicates search progress — that is, how many bits of the query have already been processed. Initially, `index` is 0. Functions used in the algorithm are

- `sub_path(string, from, to)` — returns the substring of `string` that starts at position `from` and ends at position `to`.
- `common_prefix_of(str1, str2)` — returns the common prefix of strings `str1` and `str2`.
- `get_refs(index)` — returns the list of addresses in the routing table for a prefix of length `index`.
- `random_select(refs)` — returns an address from the address list and removes it from `refs`.
- `online(ref)` — returns true if the referenced peer is online.

The algorithm first compares the common prefix of the peer's path to the query submitted. Because the first `index` bits have already been truncated from the query string, the algorithm must also adapt the peer's path. This is an optimization because at level `index` of the virtual search tree, the equality of the first `index` bits is guaranteed. Only the subsequent bits are relevant (line 3 in Figure 3) and must be compared to the query to find their common prefix (line 4). If the common path (`compath`) is as long

The Gnutella File-Sharing System (cont.)

continued from p. 60

$$2 * \sum_{i=0}^{TTL} C * (C - 1)^i = 26240$$

Recent experiments show that in a real-world setting this accumulates up to 3.5 Mbps (or 353,396 queries in 2.5 hours).[1] Caching, which reduces the traffic considerably, might remedy this problem.[1]

Because very little is known about the Gnutella network topology, one cannot estimate query duration or probability for successful search requests. Such knowledge would provide the foundation for an accurate mathematical model and would aid the development of efficient algorithms that exploit structural properties. As a first step, a recent study[2] investigated Gnutella's topology and has shown that it exhibits strong small-world properties[3] and a power law distribution of node degrees.

Free Riding and Other Social Problems

In addition to technical problems, non-technical, or social, problems such as *free riding*, which occurs when users provide no files (or few interesting files) to share, challenge Gnutella.[4] Adar and Huberman show that nearly 70 percent of Gnutella users share no files, and nearly 50 percent of all responses are returned by the top 1 percent of the sharing hosts.[4] This problem starts to transform Gnutella into a client-server-like system that might soon face technical (degradation of performance, vulnerability, and so on) and legal issues similar to Napster's.

Another social issue not addressed by Gnutella is reputation. In a P2P system, peers frequently "meet" unknown peers and have no way to judge their reputations — that is, to what extent they can trust peers and the data they provide. Gridella's P-Grid approach can also address this issue efficiently.[5]

References

1. K. Sripanidkulchai, "The Popularity of Gnutella Queries and Its Implications on Scalability," white paper, Carnegie Mellon Univ., Mar. 2001, available online at http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html.
2. M.A. Jovanovic, F.S. Annexstein, and K.A. Berman, "Scalability Issues in Large Peer-to-Peer Networks — A Case Study of Gnutella," tech. report, Univ. of Cincinnati, Laboratory for Networks and Applied Graph Theory, 2001; available online at http://www.ececs.uc.edu/~mjovanov/Research/paper.ps.
3. J. Kleinberg, "The Small-World Phenomenon: An Algorithmic Perspective," tech. report 99-1776, Cornell Univ. Computer Science Dept., Oct. 1999.
4. E. Adar and B.A. Huberman, "Free Riding on Gnutella," *First Monday*, vol. 5, no. 10, Oct. 2000; available online at http://firstmonday.org/issues/issue5_10/adar/index.html.
5. K. Aberer and Z. Despotovic, "Managing Trust in a Peer-2-Peer Information System," *Proc. 10th Int'l Conf. Information and Knowledge Management (2001 ACM CIKM)*, ACM Press, New York, Nov. 2001, pp. 310-317.


```
 1   search (peer, query, index) {
 2      found = NULL;                /* found: the address of the responsible peer */
 3      rempath = sub_path(path(peer), index+1, length(path(peer)));
 4      compath = common_prefix_of(query, rempath);
 5      IF length(compath)=length(query) OR length(compath)=length(rempath) THEN
 6           found = peer;
 7      ELSE
 8           IF length(path(peer)) > index + length(compath) THEN
 9                new_query = sub_path(query, length(compath) + 1, length(query));
10                refs = get_refs(index + length(compath) + 1);
11                WHILE |refs| > 0 AND NOT found
12                        ref = random_select(refs);
13                        IF online(ref)
14                                found = search(ref, new_query, index + length(compath));
15      RETURN found;
16   }
```

Figure 3. P-Grid search algorithm. The algorithm compares the common prefix of the peer's path to the query submitted to find the "closest" peer.

as the query or the remaining path, the peer responsible for this query is found (line 5).

Otherwise the query must be forwarded. This is only possible if the peer is sufficiently specialized (line 8). If so, it strips the common prefix off the query (line 9), queries the routing table for the list of peers to forward the query to (line 10), and forwards the remaining `new_query` recursively to a random peer from the list (if the peer is online) until the list is exhausted or the search succeeds (lines 11-14).

Constructing a P-Grid

With no global control, P-Grid construction is by local interactions only. The idea is that whenever peers meet, they refine the access structure. Peers might meet randomly because they are involved in other operations or because they systematically want to build the access structure. Assuming that by some mechanism peers meet frequently, the process works as follows.

Initially, all peers are responsible for the entire

```
1     exchange(a1, a2, r) {
2            determine the common prefix of path(a1) and path(a2) and its length lc;
3
4            exchange references at the level where the paths match;
5            li = length of remaining path of ai;
6            /* Case 1: both paths empty, introduce new level */
7            CASE l1 = 0 AND l2 = 0 AND lc < maximum possible path length
8                   extend path(a1) with 0 and path(a2) with 1;
9                   add mutual references for future search;
10           /* Case 2: one remaining path empty, split shorter path */
11           CASE l1 = 0 AND l2 > 0 AND lc < maximum possible path length
12                  extend path(a1) by one bit different to the corresp. bit in path(a2);
13                  update references of a1 with a2;
14           /* Case 3: analogous to case 2 with roles exchanged */
15                  ...
16           /* Case 4: recursively exchange with referenced peers */
17           CASE l1 > l2 > 0 AND r < maximum recursion depth
18                  take a reference from a2 at the level of the common prefix;
19                  a1 performs a new exchange with the referenced peer
20                  (which shares with a1 a longer common prefix);
21           /* Case 5: analogous to case 2 with roles exchanged */
22                        ...
23    }
```

Figure 4. P-Grid construction algorithm. When two peers meet, they divide the search space. Each takes responsibility for one half and stores the address of the other peer to cover the other half.

search space, that is, all search keys. When two peers meet, they divide the search space and each peer takes responsibility for one half and stores the other peer's address to cover the other half. The same happens whenever two peers responsible for the same path meet. If peers whose paths share a common prefix meet, they can initiate new exchanges by forwarding each other to their referenced peers. If the meeting peers' paths are in a prefix relationship, the peer with the shorter path can specialize by extending its path. To obtain a balanced P-Grid, the peer will specialize in the opposite direction from the other peer at that level. Figure 4 shows the complete construction algorithm in pseudocode.

Simulations show that this algorithm has the following properties[1]:

- The convergence speed to a complete P-Grid is (practically) independent of the total number of peers — that is, each peer participates in a constant number of exchanges independent of the population size.
- The algorithm scales gracefully as maximum path length grows.
- To obtain fast convergence, the maximum allowed recursion depth should exceed a minimum value (for example, 2 for paths of length 6).

Thus, P-Grids can be constructed efficiently in a self-organizing system without central control. Moreover, the bootstrap algorithm is uniform, self-stabilizing, and distributed.[5] Simulation results also show that the number of peers responsible for the same keys is distributed uniformly with a low deviation from the expected average number of peers responsible for a key.[1]

Data updates in the P-Grid require the updating peer to identify all replicas that store the concerned data. Simulations show that a good strategy for identifying these replicas is a breadth-first search with limited recursion breadth.[1] Still, this simple approach will not find all replicas. To compensate, the query strategy includes multiple searches on the P-Grid and chooses the correct result based on the number of identical answers. Thus, a high probability of correctness is possible.

Mapping Filenames to Binary Keys

In the P-Grid approach, we assume that search keys have a binary representation and are uniformly distributed. Neither assumption, however, holds for real filenames. We therefore provide a mapping scheme to calculate a binary representation from a filename string. To support search on these binary keys, the mapping has to satisfy a prefix property for strings s_1 and s_2:

$$s_1 \textit{ prefix } s_2 \Rightarrow \textit{key}(s_1) \textit{ prefix } \textit{key}(s_2).$$

For this mapping, we first construct a balanced trie from a sample string database. (A trie is a tree for storing strings in which there is one node for every common prefix.) Figure 5 shows this trie construction algorithm.

```
1     MakeTrie(sampledb) {
2            /* sort the sampledb in lexicographical order */
3            SortLex(sampledb);
4            /* find the length of the common prefix for all the strings in sampledb */
5            commonprefix = Length(CommonPrefix(sampledb));
6            /* We choose the median string from sampledb and take the prefix of
7                   length commonprefix+1 to split the sampledb into two
8                   approximately equal parts. It is possible to explore different
9                   alternatives at this point, in order to achieve a more balanced split */
10           if Size(sampledb) > MaxLeafStore then
11                  mid = Prefix(sampledb[Quotient(Size(sampledb), 2)],
12                         Length(commonprefix + 1);
13           for j = 1 to Size(sampledb) do
14                  if sampledb[j] is lexicographically smaller than mid then
15                         lowpart = Append(lowpart,sampledb[j]);
16                  if sampledb[j] is lexicographically greater than mid then
17                         highpart = Append(highpart,sampledb[j]);
18           if Size(lowpart) > MaxLeafStore then
19                  left = MakeTrie(lowpart)
20           else
21                  left = null;
22           if Size(highpart) > MaxLeafStore then
23                  right = MakeTrie(highpart)
24           else
25                  right = null;
26           root = mid;
27           TrieSet(root, left, right);
28    }
```

Figure 5. Trie construction algorithm. The algorithm constructs a balanced trie structure, which the mapping algorithm uses to compute binary search keys.

```
1     FindKey (trie, filename) {
2            key = {};
3            if (trie = null) or (filename is prefix or equal to trie.root) then
4                  return key;
5            else
6                  if filename is lexicographically smaller than trie.root then
7                         key = Append(key, 0);
8                         key = Append(key, FindKey(trie.left, filename);
9                  else
10                        key = Append(key, 1);
11                        key = Append(key, FindKey(trie.right, filename);
12    }
```

Figure 6. Mapping strings into binary keys. The mapping algorithm uses the trie structure to map strings to binary keys.

The trie-building algorithm takes a sample search string database `sampledb` as a parameter that contains unique strings of length `len`, which are substrings of actual search strings that were collected from the current Gnutella system. First the length of the common prefix (`commonprefix`) of all strings in `sampledb` is determined. Then the algorithm sorts `sampledb` lexicographically and selects the string at position

$$\left\lfloor \frac{Size(sampledb)}{2} \right\rfloor .$$

From this string we take the prefix of length `commonprefix+1`, which is used as the criterion to split `sampledb` into two parts of approximately equal size.

The value of this prefix is stored in the trie root and the function is called recursively for both parts to construct the left (lower) and right (higher) branch of the trie. The splitting proceeds until there are fewer than `MaxLeafStore` strings in `sampledb`.

Using this trie, we can map strings (filenames) into binary keys as shown in Figure 6.

The binary key is calculated by lexicographically comparing the string to the trie's root value. If it is prefix of or equal to the root, the calculation terminates and returns the binary key. If the string is smaller, 0 is appended to the key and the function is called recursively with the left subtree. If it is greater, 1 is appended to the key and `FindKey` is called with the right subtree.

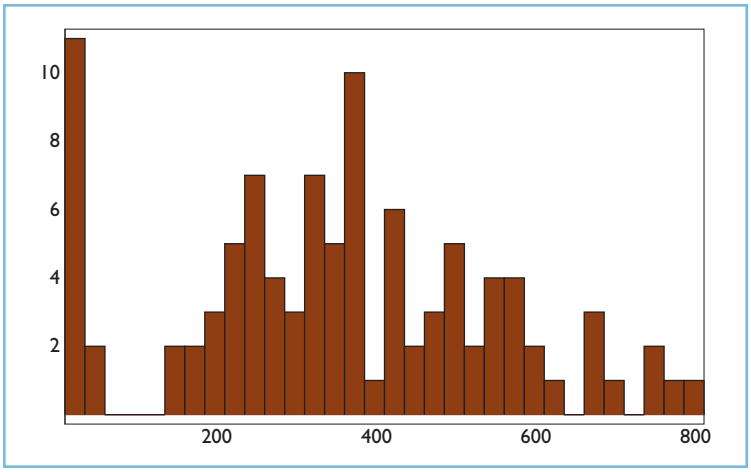

Figure 7. Classification of peers according to their workload. The workload is proportional to the number of strings (search requests) that are mapped to the key a peer is responsible for. The number of strings mapped to the same key (or peer) are classified in steps of 25; the height of each bar corresponds to the number of peers that have (approximately) the same workload.

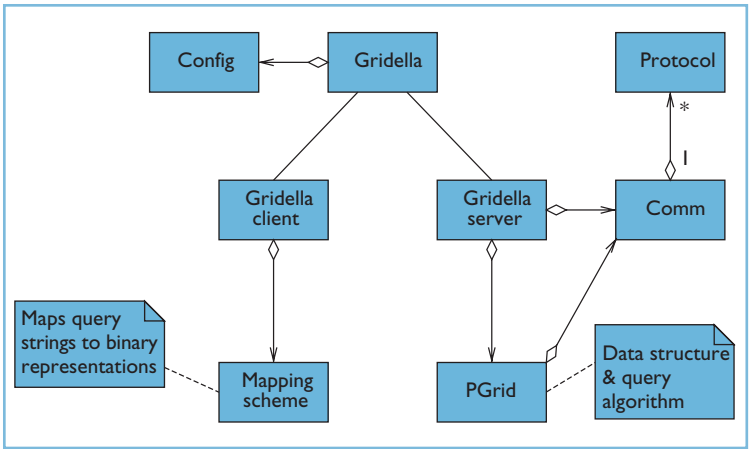

Figure 8. Gridella core system components. The Gridella client provides user-related functionality, while the server handles data management and communication.

The mapping is based on a sample database constructed by systematically collecting filenames from the current Gnutella system. We assume (and validate in experiments described later) that a large sample database effectively approximates the global distribution of filenames, such that mapping the database results in a reasonably uniform distribution of key values for all filenames. We thus package the mapping scheme and the sample database with the other P-Grid software components.

To evaluate the mapping algorithm's quality, we modified the open-source Furi (http://www.jps.net/williamw/furi/) Gnutella client to log all queries routed through it. We used this data to construct a large database of Gnutella queries, from which we derived a sample database that we used to construct the trie structure. We then tested the trie structure by encoding the complete set of search strings and analyzed the resulting key distribution.

Of 33,799 logged search strings of length 4, for example, we randomly selected 1,951 strings for the sample database. From this set we constructed the trie using `MaxLeafStore = 30`, which resulted in 99 different keys. When generating the keys for all search strings, we mapped a maximum of 798 strings to each key. A perfectly uniform distribution would result in 342 search strings per key. In the worst case, slightly more than twice the number of search strings would be encoded into the same key as with a perfectly uniform encoding. Thus, the resulting distribution is of fairly good quality with respect to uniformity, and peer workload for storing data and answering queries is distributed reasonably uniformly as well. This is also illustrated by the frequency histogram in Figure 7. Ideally, with a uniform distribution, all peers would have the same number of requests (that is, one bar at 342). In reality, we found that the majority of peers has similar workloads, and only few have high workloads or are not loaded at all.

Gridella System

Gridella is our P-Grid-based, Gnutella-compatible P2P system. It targets efficient distributed search, network traffic reduction, component reuse, and multiprotocol support. Gridella is written in Java and will be released in spring 2002 under the GNU general public license.

Components and Communication Model

Figure 8 shows the main components of the Gridella system. Locally (that is, on a node) the implementation of a Gridella peer follows a client-server architecture. The Gridella client provides all user-related functionality so that the user can use an arbitrary GUI while the Gridella server handles data management and communication. The communication subsystem provides communication abstractions for including arbitrary protocols (we currently support the Gnutella and Gridella protocols), which allows Gridella to interoperate with other P2P systems.

Figure 9 shows a typical search interaction between two Gridella peers. When a user enters a query, the local Gridella client maps the query into the binary P-Grid representation and sends it to the local Gridella server, which checks whether it is responsible for the request. If yes, it returns the data; otherwise, it determines which

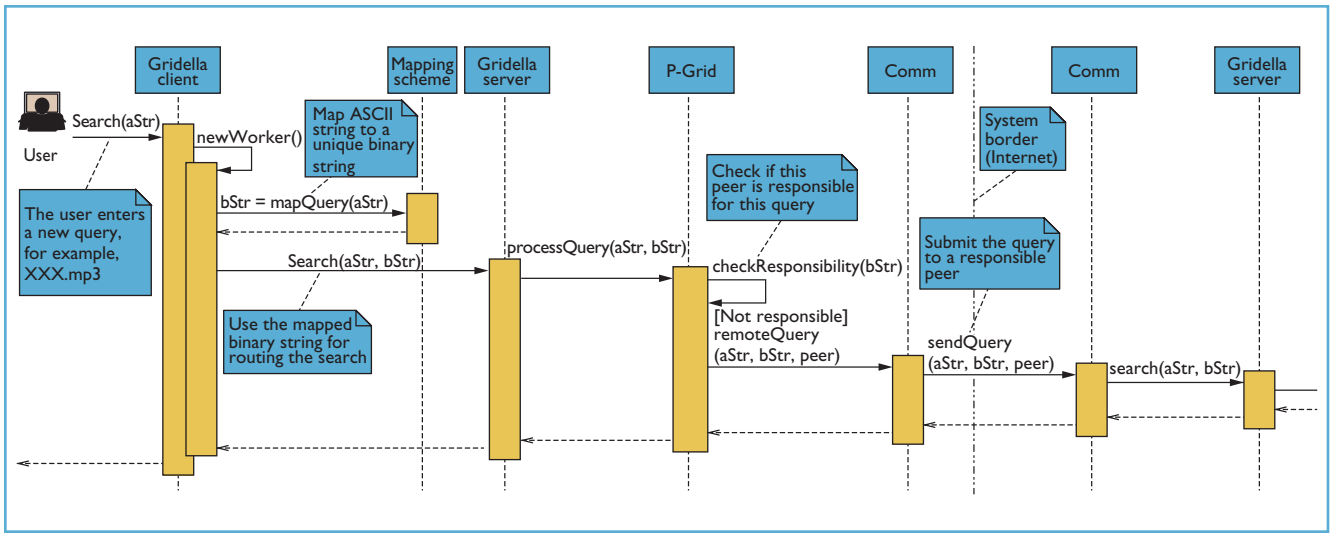

Figure 9. A Gridella interaction. Queries are mapped into binary keys and sent to the local Gridella server, which either answers the query or forwards it to the appropriate peer.

peer to contact and forwards the query using the communication subsystem. The peer receiving the query forwards it to its local Gridella server and the pattern repeats.

Other interactions, such as meetings between two peers, announcing a peer's availability, or downloading data, are straightforward. Interacting with non-Gridella peers is simple as well. If the recipient is a Gnutella peer, for example, the Gridella peer skips the mapping step and forwards the original query via the Gnutella protocol.

Gridella vs. Gnutella: Performance Comparison

Gridella can be viewed as a layer on top of Gnutella that provides additional abstractions and functionality: Gnutella supplies the basic communication and Gridella adds directed, efficient search through its underlying P-Grid approach. In our study, we compared the number of messages required to locate a specific data item in Gridella and Gnutella with success probability of 0.99. We considered populations of 20,000 to 200,000 peers with a 0.3 probability of their being online. Table 1 shows the results of our comparison.

To achieve our search goal in Gnutella, it was sufficient to create 22 replicas of each data item and to have a search horizon of 70 percent of all Gnutella peers (that is, to reach 70 percent of the Gnutella population with a search message). We then computed the required number of search messages (using the formula discussed in the sidebar, "The Gnutella File-Sharing System").

For Gridella, we assumed that each peer stores 1,000 data items, and then determined the number of messages required to traverse the system's search structure in the worst case. The variations listed in Table 1 for Gridella result from the simulation of the probabilistic process modeling Gridella's search algorithm. The steps in the results for Gnutella correspond to the steps in the TTL, which must be incremented for higher numbers of peers to meet the 70 percent requirement. The results clearly demonstrate the benefit of using an access structure, even if we take into account some modest storage demand and update overhead in Gridella.

Table 1. Performance comparison of Gridella and Gnutella.

Peers	Gridella messages	Gnutella messages
20,000	61	8,744
40,000	63	26,240
60,000	65	26,240
80,000	65	78,728
100,000	68	78,728
120,000	69	78,728
140,000	68	78,728
160,000	69	78,728
180,000	69	78,728
200,000	72	78,728

Future Work

We plan to introduce reputation and trust into Gridella based on an approach that uses P-Grid for decentralized storage of reputation data.[6] The nat-

Related Work in Decentralized Systems

Several approaches address scalable, decentralized access schemes. Unlike our approach, these consider only exact search for file (or object) identifiers rather than text-based search for filenames.

Several serverless distributed file systems, such as Farsite,[1] Freenet,[2] or OceanStore,[3] provide secure file storage in an insecure environment. All encrypt the information to be stored and devise mechanisms for distributing and retrieving files in an unreliable environment. Like Gridella, they require a self-organizing routing structure, and, as published, their routing approaches are similar to P-Grid. Unlike Gridella, they do not focus on information sharing and search.

Several architectures for decentralized lookup services propose scalable access structures to search for identifiers stored in distributed directories.[4-6] Plaxton et al.'s approach, in particular, uses virtual binary search trees similar to P-Grid.[5] These approaches differ in two respects from our work:

- They consider only the problem of equality search for keys.
- Some of their assumptions mildly violate peer autonomy. For example, they assume fixed roles for peers that are globally assigned, either by virtue of IP addresses[4] or by a globally shared identification of peers.[5] Others require a shared list of bootstrap entries to the network[6] — a situation the P-Grid bootstrap algorithm avoids.

In addition to these approaches, ongoing work seeks to increase the interoperability of P2P systems and to define a universal architecture for them. JXTA defines a three-layer P2P software architecture, a set of XML-based protocols, and a number of abstractions and concepts such as peer groups, pipes, and advertisements to provide a uniform platform for applications using P2P technology and to allow P2P systems to interact.[7] We are evaluating JXTA for Gridella.

References

1. W.J. Bolosky et al., "Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs," *Proc. Int'l Conf. Measurement and Modeling of Computer Systems* (SIGMETRICS 2000), ACM Press, New York, June 2001, pp. 34-43.
2. I. Clarke et al., "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Proc. Designing Privacy Enhancing Technologies: Int'l Workshop Design Issues in Anonymity and Unobservability*, LNCS 2009, Springer-Verlag, Heidelberg, Germany, Feb. 2001, pp. 46-66.
3. S. Rhea et al., "Maintenance-free Global Data Storage," *IEEE Internet Computing*, vol. 5, no. 5, Sept./Oct. 2001, pp. 40-49.
4. I. Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM*, ACM Press, New York, Aug. 2001, pp. 149-160.
5. C.G. Plaxton, R. Rajaraman, and A.W. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," *Proc. ACM Symp. Parallel Algorithms and Architectures*, ACM Press, New York, June 1997.
6. S. Ratnasamy et al., "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM*, ACM Press, New York, Aug. 2001, pp. 161-172.
7. L. Gong, "JXTA: A Network Programming Environment," *IEEE Internet Computing*, vol. 5, no. 3, May/June 2001, pp. 88-95.

ural next step is to address security issues such as authenticity and confidentiality. These improvements would make P2P an interesting environment for new e-commerce models.

We also intend to address free riding by introducing economic concepts to force users to pay for the services they use. Such a market-driven approach does not necessarily mean monetary exchange, however. We could implement a micropayment system with an artificial currency to balance requests with offers.[7,8] Offers and downloads from a peer would earn credits, which could be used to pay for services requested.

Conclusion

The World-Wide Web proved the Internet community's ability to incubate revolutionary systems and somewhat "out-perform" the scientific community. By popularizing the P2P approach in simple yet very successful and influential systems such as Napster and Gnutella, the Internet community has proven this ability again. Although such developments help spread and advance new technologies, when they lack a scientific foundation, long-term development is impeded.

We still can give P2P systems firm scientific foundations by combining state-of-the-art methodological and engineering know-how. The work presented in this article is a first step. Considerable research and experimentation remains to make P2P systems feasible for application domains beyond mere MP3 and image exchange, such as creating a new paradigm for decentralized e-commerce systems, or for new types of network infrastructures such as mobile ad hoc networks.[9]

References

1. K. Aberer, "P-Grid: A Self-Organizing Access Structure for P2P Information Systems," *Proc. Int'l Conf. Cooperative Information Systems*, Lecture Notes in Computer Science 2172, Springer-Verlag, Heidelberg, Germany, 2001, pp. 179-194.
2. T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Data on Air: Organization and Access," *IEEE Trans. Knowledge and Data Eng.*, vol. 9, no. 3, May/June 1997, pp. 353-372.
3. C.G. Plaxton, R. Rajaraman, and A.W. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," *Proc. ACM Symp. Parallel Algorithms and*

Architectures, ACM Press, New York, 1997, pp. 311-320.
4. H. Yokota, Y. Kanemasa, and J. Miyazaki, "Fat-Btree: An Update-Conscious Parallel Directory Structure," *Proc. 15th Int'l Conf. Data Eng. (ICDE)*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1999, pp. 448-457.
5. M. Schneider, "Self-Stabilization," *ACM Computing Surveys*, vol. 25, no. 1, Mar. 1993, pp. 45-67.
6. K. Aberer and Z. Despotovic, "Managing Trust in a Peer-2-Peer Information System," *Proc. 10th Int'l Conf. Information and Knowledge Management* (2001 ACM CIKM), ACM Press, New York, 2001, pp. 310-317.
7. Mojo Nation Technology Overview, 14 Feb. 2000, http://www.mojonation.net/docs/technical_overview.shtml.
8. M. Stonebraker et al., "Mariposa: A Wide-Area Distributed Database System," *VLDB J.*, vol. 5, no. 1, 1996, pp. 48-63.
9. J.P. Hubaux et al., "Towards Self-Organized Mobile Ad hoc Networks: the Terminodes Project," *IEEE Comm.*, vol. 39, no. 1, Jan. 2001, pp. 118-124.

Karl Aberer is a full professor at the Swiss Federal Institute of Technology Lausanne (EPFL), where he heads the Distributed Information Systems Laboratory. His main research interests are information management for the information economy and the self-organization of information systems. He received a PhD and a diploma, both in mathematics, from ETH Zurich, Switzerland.

Manfred Hauswirth is an assistant professor at the Technical University of Vienna. He received an MSc and a PhD in computer science from TU Vienna. His research focuses on distributed systems, e-commerce, and Internet applications. He is the principal researcher in the Minstrel project, a senior researcher in the Opelix EU project, and, beginning in 2002, will be a member of Karl Aberer's group.

Magdalena Punceva is a PhD student at the Swiss Federal Institute of Technology Lausanne (EPFL). She received a BSc from the Faculty of Electrical Engineering, University of Saints Cyril and Methodius in Skopje. She attended the graduate school in computer science at EPFL. Her research interests include peer-to-peer information systems, decentralized access structures, databases, mobile agents, and economic and bio-inspired principles.

Roman Schmidt is a graduate student at the Technical University of Vienna. He is currently working on a master's thesis about the Gridella peer-to-peer system. His research interests include distributed systems, e-commerce, Internet applications, and real-time systems.

Readers can contact the authors at {karl.aberer, magdalena.punceva}@epfl.ch or {M.Hauswirth, R.Schmidt}@infosys.tuwien.ac.at.