Experimental evaluation results of an advanced DHT-based peer-to-peer system*

Karl Aberer, Manfred Hauswirth, Roman Schmidt

Swiss Federal Institute of Technology Lausanne (EPFL)
Distributed Information Systems Laboratory
1015 Lausanne, Switzerland
{Karl.Aberer, Manfred.Hauswirth, Roman.Schmidt}@epfl.ch

Abstract

P-Grid is a decentralized DHT-based peer-to-peer system with logarithmic search complexity. As it is intended to provide a platform for distributed information management beyond mere file-sharing it supports an update functionality with lazy consistency guarantees, identity management disentangling the DHT from the underlying networking infrastructure, and decentralized load balancing. In this paper we first give a brief overview of the theoretical foundations of P-Grid and the supporting functionalities. Then we highlight some interesting implementation details and present briefly some experimental evaluation results. To our knowledge this is one of the first reports on experimental evaluation results of a structured peer-to-peer system in a real-world networking environment.

1 Introduction

P-Grid is a distributed data structure based on the principles of distributed hash tables (DHT) [7]. As any DHT approach P-Grid is based on the idea of associating peers with data keys from a key space. Without constraining general applicability we will only consider binary keys in the following. In contrast to other DHT approaches we do not impose a fixed or maximal length on the keys.

*The work presented in this paper was supported (in part) by the European Commission under contract FP6-507483, project DIP (Data, Information, and Process Integration with Semantic Web Services).

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

In P-Grid peers refer to a common underlying tree structure in order to organize their routing tables. In the following we will assume that the tree is binary. This is not a fundamental limitation as a generalization of P-Grid to k-ary structures has been introduced in [4], but will simplify the presentation. Note that the underlying tree does not have to be balanced but may be of arbitrary shape, thus facilitating to adapt the overlay network to unbalanced data distribution [1].

Each peer $p \in P$ is associated with a leaf of the binary tree. Each leaf corresponds to a binary string $\pi \in \Pi$. Thus each peer p is associated with a path $\pi(p)$. For search, the peer stores for each prefix $\pi(p,l)$ of $\pi(p)$ of length l a set of references $\rho(p,l)$ to peers q with property $\overline{\pi(p,l)} = \pi(q,l)$, where $\overline{\pi}$ is the binary string π with the last bit inverted. This means that at each level of the tree the peer has references to some other peers that do not pertain to the peer's subtree at that level. Multiple peers are required for each level to compensate for failing peers. This enables the implementation of prefix routing for search.

Each peer stores a set of data items $\delta(p)$. Ideally for $d \in \delta(p)$ the key $\kappa(d)$ of d has $\pi(p)$ as prefix. However, we do not exclude that temporarily also other data items are stored at a peer, that is, the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. In addition, peers also maintain references $\sigma(p)$ to peers having the same path, i.e., their replicas.

In a stable state the set of paths of all peers is prefix-free and complete, i.e., no two peers p and q exist such that $\pi(p) \subset \pi(q)$, i.e., $\pi(p)$ is a proper prefix of $\pi(q)$ and if there exists a peer p with path $\pi(p)$, then there also exists a peer q with $\overline{\pi(p)} = \pi(q)$. This guarantees full coverage of the search space and complete partitioning of the search space among the peers. All data stored at a peer then matches its path.

1.1 Search in P-Grid

P-Grid uses a prefix routing strategy for search. In the following, the notation $p\ ?\kappa\ q$ means peer p queries peer q for key κ. To search for a key κ starting at peer

q the following forwarding algorithm is used:

Upon receiving the event $p\ ?\kappa\ q$ from p, peer q checks whether its path is a prefix of κ. If yes, it checks whether it can return a query result from its data store. If not, it selects a peer r having a prefix of maximal length with κ from its routing table and issues a query event $q\ ?\kappa\ r$, i.e., the search continues at r.

The algorithm always terminates successfully: Due to the definition of $\rho(p, l)$, this prefix routing strategy will always find the location of a peer at which the search can continue (use of completeness) and each time the query is forwarded, the length of the common prefix of $\pi(p)$ and κ increases. It is obvious that this search algorithm is efficient ($O(log(|\Pi|))$) for a balanced tree, i.e., all paths associated with peers are of equal length. In [1] we show that due to the probabilistic nature of the P-Grid approach this does not pose a problem. The expected search cost measured by the number of messages required to perform the search remains logarithmic, independently how the P-Grid is structured.

2 Implementation

P-Grid is implemented in Java using an XML-based protocol with compression. The implementation is based on the algorithms presented in [2] to construct and maintain a storage and replication load-balanced tree, i.e., peers store approximately the same amount of data keys and data keys are stored (replicated) approximately equally often at peers.

P-Grid's XML-based protocol is defined in [9]. We decided to base the P-Grid protocol on XML to enable interoperability with other implementations. The main drawback, however, is the overhead incurred by XML which causes large message sizes which may lead to high bandwidth consumption. Thus the protocol is not exchange "in-the-clear" but uses ZLIB compression to reduce bandwidth consumption to about 3-5% of the uncompressed size. Although this increases the CPU usage for receiving and sending messages, bandwidth consumption is the more critical resource.

The protocol consists of 6 message types: *Bootstrap*, *Exchange*, *Query*, *QueryReply*, *SearchPath*, and *SearchPathReply*. The Bootstrap message is used by peers joining a P-Grid the first time to learn about some other participating peers. Exchange messages are used to construct and maintain a P-Grid. Query messages represent search requests and QueryReply messages are their responses containing the found data. SearchPath and SearchPathReply are required for load balancing. SearchPath is sent if a peer wants to become another peer's replica and SearchPathReply is returned if a peer suitable for replication is found. The protocol also defines the handshake between two peers after a connection was established. The message sizes vary from a few bytes for Bootstrap, Query, and SearchPath messages to more than 100 kb for Exchange, QueryReply, and SearchPathReply messages depending on the amount of stored data keys of a peer respectively the amount of matching data keys for a query.

During P-Grid construction and maintenance data key sets may have to be exchanged frequently between two peers and can be of considerable size depending on the amount of data a peer stores. Thus the protocol uses algebraic signatures [5, 6] identifying equal data key sets at peers and thus reduces network traffic. Data key set signatures reduce the amount of transmitted data keys to about 70% of the original amount. The currently implemented (simple) approach of transmitting the complete sets if a difference is detected will be improved in the next version of the software. We will partition the data sets and use algebraic signatures on the subsets to focus the detection of changes and thus minimize the amount of information that needs to be exchanged.

The implementation is structured into two parts: The P-Grid library provides all functionality required to construct and maintain a P-Grid, and also implements the search algorithm. The P-Grid application layer uses this library and implements a file sharing application for performing tests and experiments with a graphical user interface. The library is intended for developers who want to use P-Grid as middleware and is well documented with usage examples and Java API documentation. The file-sharing application (Gridella) is intended as an example to support developers to implement their own applications and demonstrates the capabilities of P-Grid. Gridella shares files and enables full-text search on file names. Matching files are presented via the GUI and can be downloaded. Full-text search is achieved by indexing suffixes. Although this may seem inefficient at first glance, P-Grid is powerful enough to deal even with a high number of inserts. The implementation and the source code of P-Grid and Gridella are available for Java platforms at http://www.p-grid.org/.

3 Experimental results

This section presents some of the the results of our experiments with the P-Grid implementation. The experiments aimed at validating the mathematical results presented in [2] in a real-world setting and to test the quality, correctness, and robustness of the protocol. A further motivation was to compare P-Grid with other implemented P2P systems such as Gnutella, especially in respect to bandwidth consumption as the most critical resource.

3.1 Setup

The experiments were performed with 32 peers running on 16 Linux computers in a 100 MBit local area network. Each peer shared 100 files resulting in 1700–1900 data items (depending on the file names) initially managed by each peer. Data items represent shared files including a data key generated from the file name

or from a suffix of the file name. Peers were willing to manage at most $2 * m_{store} = 10000$ data items, the maximum number of recursive exchanges *recmax* was 3, and the probability for splitting the tree p_{split} was 0.1. The split probability influences the speed of constructing a P-Grid. $p_{split} = 0.1$ means that a path is split with a probability of 10% only if all other requirements for splitting (sufficient number of data items, etc.) are fulfilled. Peers store a maximum of 10 fidget peers (*fidgetmax*)—used for introducing additional randomization— and 10 references for each level of the P-Grid tree (*refmax*) in their routing tables.

The parameters bl and $prob_c$ influence the load balancing process to avoid very expensive oscillatory behavior in terms of maintenance without making the process too slow. In the experiment we used $bl = 0.1$ and $prob_c = 0.25$ so that peers have to gather at least 10 statistic entries per level (*minchange*) before initiating load-balancing.

The experiment ran between one and several days with an average *startup period* between 2 and 3 hours. The startup period is the time to build-up a P-Grid from scratch. At the beginning peers have no path, manage only their shared files, and only know one peer for bootstrapping. At the end of the startup period the paths of the peers have stabilized and the *operation period* of the experiment starts. Peers were supposed to be online all the time to avoid additional efforts incurred by compensating for unavailable peers (this was done in another experiment).

3.2 P-Grid construction

The construction algorithm tries to achieve three goals: (a) a balanced binary tree, (b) balanced replication factor of paths and data keys, and (c) balanced distribution of data keys among peers.

A binary tree resulting from one of our experiments is shown in Figure 1. The number of peers per path is given in the nodes of the tree and the number of peers responsible for a path prefix is given at the edges of the tree.

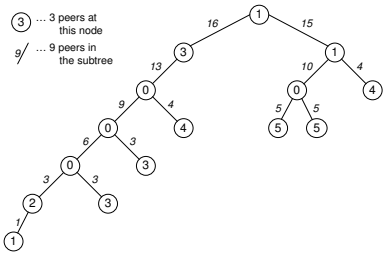

Figure 1: A resulting virtual binary P-Grid tree

The tree seems to be unbalanced as the tree depth is unequal for some paths, e.g., the maximum tree depth for '0*' is 6 whereas the depth for '1*' is 3, and the number of peers per path differs (e.g., 10 peers for path '10*' and only 4 peers for path '11*'). This imbalance is due to the uneven distribution of data keys in the experiment. About 65% of the data item keys began with '0' and more than 80% of these data items have a key prefix of '00' leading to the longer path for '00*'. A similar data key distribution is responsible for the unbalance of path '10*' and '11'. Achieving a tree of this shape, however, was intended to balance memory and replication loads. Despite this unbalanced tree shape, search costs remain logarithmic as shown in [1]. Informally, the reason is that path resolution does not work bit-by-bit but may resolve longer bit strings at one peer thus on average providing logarithmic search complexity.

The tree also is not prefix-free: 1 had no path, 3 had path '0', 2 peers had path '00000', and 1 peer had path '1'. This also was expected because due to the choice of parameter m_{store} the amount of data items in the P-Grid was not high enough to create a prefix-free tree (peers have to manage more than m_{store} data items to split their path and specialize). This negative effect of an "under-loaded" P-Grid may cause that some data keys may not be found. However, our experiments and simulations have shown that this can easily be compensated by repeating a query two times starting at a random peer selected from the fidget list.

In Figure 1 the replication factor of peers responsible for paths is between 3 and 5. The peer responsible for path '000000' is replicated by the two peers at path '00000' and the peers at paths with prefix '0' could still be replicated by the peers at path '0' if further data items become available. The average replication factor for data items is between 4 and 5. Each peers manages between 7000 and 9500 (initially 1700–1900 data items). This means the average storage load is approx. 80%.

This and other construction experiments performed validated our theoretical results.

3.3 Search performance

For testing search performance we generated search requests at random peers and analyzed the results. The names we inserted have a common substring '-az-' and the same file extension '.pdf' enabling searching for all files in P-Grid. Unfortunately this also leads to the unbalanced distribution of data keys. However, this can also occur in real-world settings. Searches were initiated during the operation time with all possible suffixes of the common substrings to generate different search keys processed by different peers. The search results always provided the complete result set with the expected logarithmic number of messages to complete the requests.

3.4 Bandwidth consumption

Network bandwidth consumption was measured separately for the startup phase and the stable operation phase. The amount of shared files was constant and the measurements do not include any traffic arising

from search requests and their responses as we wanted to find out about the maintenance "noise" of the implementation.

During P-Grid construction the overall network bandwidth usage is about 60–70kb/sec leading to an average bandwidth of about 2.0 kb/sec per peer. Depending on the path of a peer the minimum bandwidth consumption of a peer is about 300 bytes/sec and the maximum is approximately 3.0 kb/sec. This network traffic is fairly low and can easily be tolerated. If compression was not used the overall network bandwidth consumption would have been 2.0–3.0 Mb/sec and the average peer bandwidth consumption would have been approx. 80.0 kb/sec. The experiment shows that protocol compression is highly efficient and reduces the network bandwidth consumption down to 2.5%.

During the operation of P-Grid the network bandwidth consumption is significantly smaller because peers are not changing their paths anymore and most of the data items are already at the responsible peers. The overall P-Grid maintenance traffic using the compressed protocol is about 11–12 kb/sec meaning an average bandwidth consumption of about 350 bytes/sec per peer. The minimum bandwidth consumed was approx. 150 bytes/sec and the maximum was 1.5 kb/sec. The bandwidth consumption is reduced down to approx. 4% by compression.

To put these results in perspective we compare them with Gnutella. To our knowledge no implementation-based results exist for other DHTs that we could use for comparison.

Most of the available analyses of Gnutella network traffic [8, 10] are based on the original protocol. The bandwidth consumption varies from 6 kBit/sec per connection [8] up to 3.5 MBit/sec for a single peer [10]. A bandwidth consumption of 6 kBit/sec per connection results in a bandwidth consumption of 3 kb/sec per peer assuming 4 open connections, i.e., the bandwidth consumption of a Gnutella peer is between 3.0 and 437.5 kb/sec for network maintenance and searches. Compared to P-Grid's startup period the bandwidth consumption of Gnutella is about 1.5-218 times higher and about 8.5-1250 times higher during normal operation. [8] showed that Gnutella's maintenance traffic can make up more than 50% of the overall network traffic, i.e., 1.5-218.8 kb/sec. Compared to P-Grid's maintenance traffic during normal operation the bandwidth consumption of Gnutella is approximately 4.2-625 times higher.

4 Conclusions

The practical experiments we did with our P-Grid implementation validated our theoretical findings and allowed us to address some of the problems, for example, protocol message size, that can only be assessed in detail when theory is put into practice. Although the theoretical foundations of P2P systems are developing rapidly only very few real-world measurements and experiments exist. To our knowledge no other experimental implementation-based study for DHTs exists so far. Experiments, however, are necessary since in our opinion a number of assumptions underlying current research do not hold in a real-world network, for example, the problems caused by changing addresses/identification as discussed in [3]. Our next steps in the development of P-Grid will be the addition of semantic search (currently under development) and further experiments to include query traffic and loads. Our long-term goal is to provide a fully functional middleware layer for distributed information systems.

References

[1] Karl Aberer. Efficient Search in Unbalanced, Randomized Peer-To-Peer Search Trees. Technical Report IC/2002/79, EPFL, 2002. http://www.p-grid.org/Papers/TR-IC-2002-79.pdf.

[2] Karl Aberer, Anwitaman Datta, and Manfred Hauswirth. The Quest for Balancing Peer Load in Structured Peer-to-Peer Systems. Technical Report IC/2003/32, EPFL, 2003.

[3] Karl Aberer, Anwitaman Datta, and Manfred Hauswirth. Efficient, self-contained handling of identity in Peer-to-Peer systems. *TKDE*, 2004. To appear.

[4] Karl Aberer and Magdalena Punceva. Efficient Search in Structured Peer-to-Peer Systems: Binary v.s. k-ary Unbalanced Tree Structures. In *DBISP2P*, 2003.

[5] W. Litwin, R. Mokadem, and S.J. T. Schwarz. Disk Backup Through Algebraic Signatures in Scalable and Distributed Data Structures. In *WDAS*, 2003.

[6] W. Litwin and S.J. T. Schwarz. Algebraic Signatures for Scalable Distributed Data Structures. Technical report, CERIA, 2002. http://ceria.dauphine.fr/FileSignatureCalculationforSDDS-thomas2.pdf.

[7] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *SPAA*, 1997.

[8] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems. *IEEE Internet Computing*, 6(1), 2002. http://people.cs.uchicago.edu/~matei/PAPERS/ic.pdf.

[9] Roman Schmidt. The P-Grid protocol. Technical report, EPFL, 2004.

[10] K. Sripanidkulchai. The Popularity of Gnutella Queries and Its Implications on Scalability, March 2001. http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html.