P-Grid : A Self-Organizing Access Structure for P2P-Informationsystems

Xiaoli Zhang

11th December 2004

Wintersemester 2004/05
Proseminar : P2P Informationsystems
Professor : Prof. Weikum
Tutor : Josiane Parreira

Contents

Part I
P-Grid : A Next-generation P2P Informationsystem

1 Introduction

1.1 What is P-Grid ?

Peer-To-Peer (P2P) systems are driving a major paradigm shift in the era of genuinely distributed computing. But 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.

1.2 What is so special about it?

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 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.

In this article we present *Peer-Grid (P-Grid)* , which draws on research in distributed and cooperative information systems to provide a *decentralized, scalable data access structure*. And P- Grid is a *next generation peer-to-peer platform* for distributed information management file-sharing.

2 P-Grid Distributed Search Structure

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

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

P-Grid can be imagined as a *virtual binary search tree* . It distributes replication over a community of peers. Also it supports a search which is very effective - which means, search time and number of generated messages grow in the "running time" $O(\log(2n))$ with the number of data items n in the network. Unlike peers in other approaches that construct scalable, tree-based, distributed indexing structures; in an unreliable environment, 2-4 peers (like 2-4-trees) in P-Grid perform construction and operations, like a search or an update, without any

central control or global knowledge.

As a resource allocation problem, the load balancing is critical to support *availability, accessibility, high scalability and throughput.* If the load balancing would be poor, it may in fact gradually transform a P2P system into a backbone-based system as it was observed for *Gnutella* . For systems supporting equality-based lookup of data only, a solution of the problem of non-uniform workloads may be to apply hash functions to the data keys, thus uniformly distributing workload, both for storage and query answering.

In combination with using balanced search structures, i.e., balanced distributed search trees, this approach leads to uniform load distribution among the participating peers. However, it is limited if further semantics of the data keys is exploited, for example, in the simplest case when the ordering of data keys is used to support prefix or range queries. This is critical for DB-oriented applications.

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.

2.1 Example for P-Grid

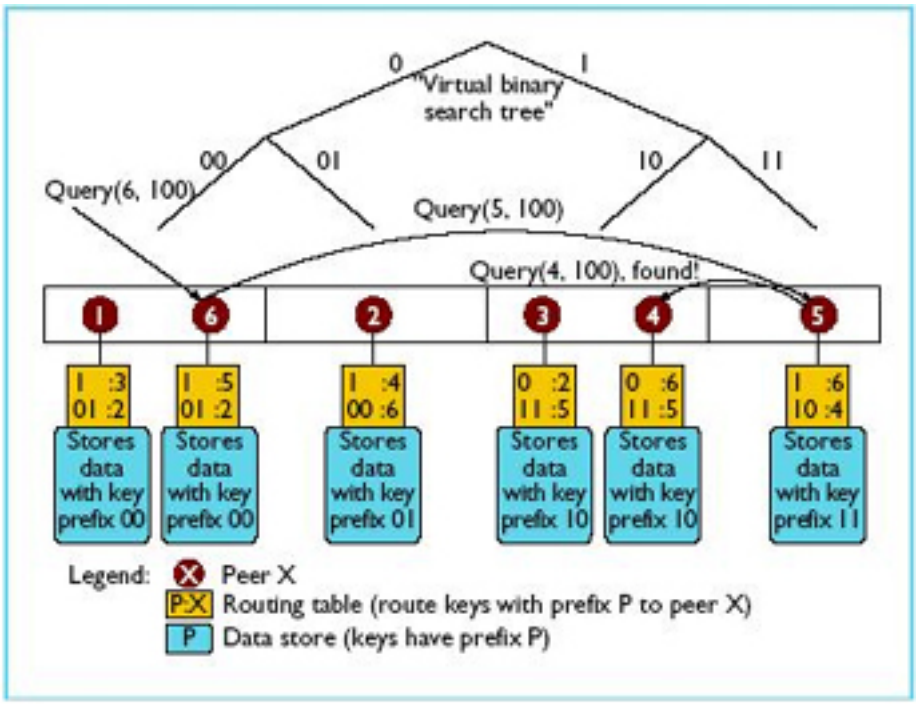


Figure 1 : Example P-Grid: Each peer is responsible for part of the overall tree. When a peer receives a query it cannot answer, it refers to its routing table to find the appropriate peer to forward the request to.

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 partitions 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.

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.

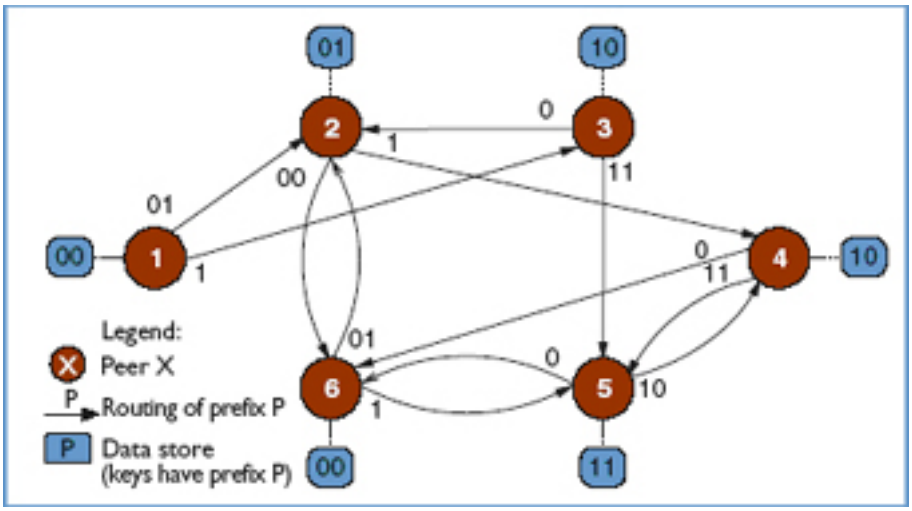


Figure 2 : Example P-Grid network. Peer routing tables provide at least one path from any peer receiving a request to one of the peers holding a replica so that any query can be satisfied regardless of the peer queried.

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 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.

In contrast, we use an *adaptive, self organizing mechanism* to globally balance data replication. Different to storage load, *peers cannot locally detect non-uniform replication of data in the entire network* . We employ a sampling-based method to detect imbalance and to dynamically adapt replication. Thus data will be dynamically replicated while peers aim at using their storage capacity optimally. An important aspect is the mutual dependency among storage load balancing and uniform replication.

When peers attempt to locally balance their storage load they may compromise globally uniform replication. By simulation we show for our approach that the system converges to a state where both load balancing goals are achieved in combination. This reactive load balancing of replication factor in a self-organized manner is possible in P-Grid *without affecting the structural properties of the system* because of the independence of peer identifier and data (keys) associated with the peer.

2.2 First Little Conclusion

We have swn with P-Grid that *self organization principles can also be applied to structured P2P systems*. Instead of the situation of unstructured systems, where search algorithms are designed in order to take advantage of the emergent overlay network structures, we are going to design the self-organization process *to gather to an overlay network* such that demonstrably efficient search algorithms can be applied and at the same time load balancing goals are achieved.

2.3 The P-Grid Search Algorithm

```
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.

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 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.

3 Important Applications of P-Grid

3.1 Updates in P-Grid

P2P systems were, until recently, primarily used for sharing static, read-only files. Therefore most P2P systems didn't provide update mechanisms that would work in the presence of replication. For example, centralized (or hierarchical) P2P systems, such as *Napster* or *Fast Track*, maintain a centralized index of data items, which are available at online peers. If an update of a data item occurs this means that the peer that holds the item changes it. Subsequent requests would get the new version.

However, updates are not propagated to other peers which replicate the item. As a result, multiple versions may coexist under the same identifier. The same holds true for most decentralized systems such as *Gnutella*.

Some systems partially make updates, what, for example, exists in *Freenet* . An update here, is routed "downstream" based on a key-closeness relation. Since the network may change during the perform of this action and no precautions are taken to notify peers that come online after an update has occurred, consistency guarantees are limited.

3.2 Handling Dynamic Addresses and Identify the Peers

As IP addresses have become a scarce resource, most computers on the Internet no longer have permanent ip-addresses. For client computers this is usually not a big problem but with the advent of P2P systems, where every computer acts both as a client and as a server, this has become problem, which increases.

In advanced P2P systems ad-hoc connections to peers have to be established, which can only be done if the receiving peer has a permanent IP address. To solve this case we have designed a service based on P-Grid, which is a completely decentralized, self-maintaining, lightweight, and sufficiently secure peer identification service. This service allows us to consistently map unique peer identifiers, in particular the logical identity of peers used for routing in P-Grid, onto dynamic IP addresses. It is designed to operate in environments with low availability of the peers.

The basic idea is to store the mappings in P-Grid itself: Peers store their current id/IP mapping in P-Grid and update it if the IP address changes (for example, if they come online again). For routing search requests while searching id/IP mappings using P-Grid's routing infrastructure, peers use cached id/IP mappings. If these cached entries are stale, they are updated by recursively querying the P-Grid again.

This may look as an unsolvable, recursive "hen-egg problem", but we demonstrate that not only most of the original queries will be answered successfully, but also, that the recursions triggered by failures will lead to a partial "self-healing" (a different form of self organization) of the whole system by updating the caches.

On the part of the security, we apply a combination of PGP like public key distribution and a quorum-based query scheme. These public keys themselves are stored in P-Grid, and replication can provide guarantees that are probabilistically analogous to PGP's web of trust. The approach can easily be adapted to other application domains, i.e., be used for other name services, because we don't impose any constraints on the type of mappings.

Motivated by the problem of handling peer identity in a setting where peers' physical addresses change because of network dynamics we thus achieved a self-contained and self-maintaining directory service for P-Grid.

4 P-Grid Construction

There is *no global control*, only by local interactions, and the peers meet randomly. Initially, all peers are responsible for all the search keys.

When two peers meet, they decide to split the search space into two parts and take over responsibility for one half each, also store the reference to other peer. The same happens whenever two peers meet, that are responsible for the same interval at the same level.

However, as soon as the P-Grid develops, also other cases occur. Namely, peers will meet:

1. their keys share a common prefix

2. their keys are in a prefix relationship

These considerations give rise to the following algorithm that two peers a1 and a2 execute when they meet.

4.1 What Happens if two Peers meet?

```
exchange(a1, a2, r)
{
commonpath = common_prefix_of(path(a1), path(a2));
lc = length(commonpath);
IF lc > 0
(* exchange references at the level where the paths agree *)
commonrefs = union(refs(lc, a1), refs(lc, a2));
refs(lc, a1) = random_select(refmax, commonrefs);
refs(lc, a2) = random_select(refmax, commonrefs);
l1 = length(sub_path(path(a1), lc + 1, length(path(a1))));
l2 = length(sub_path(path(a2), lc + 1, length(path(a2))));
(* Case 1:  if both remaining paths are empty introduce a new level
*)
CASE l1 = 0 AND l2 = 0 AND length(commonpath) < maxlength
path(a1) = append(path(a1), 0);
path(a2) = append(path(a2), 1);
refs(lc + 1, a1) = a2;
refs(lc + 1, a2) = a1;
(* Case 2:  if one remaining path is empty split the shorter path *)
CASE l1 = 0 AND l2 > 0 AND length(commonpath) < maxlength
path(a1) = append(path(a1), value(lc+1, path(a2))^ -;
refs(lc + 1, a1) = a2; refs(lc + 1, a2) = random_select(refmax,
union(a1, refs(lc+1, a2));
(* Case 3:  analogous to case 2 *)
CASE l1 > 0 AND l2 = 0 AND length(commonpath) < maxlength
5
path(a2) = append(path(a2), value(lc+1, path(a1))^ -;
refs(lc + 1, a2) = a1;
refs(lc + 1, a1) = random_select(refmax, union(a2, refs(lc+1, a1));
(* Case 4:  recursively perform exchange with referenced peers *)
CASE l1 > 0 AND l2 > 0 AND r < recmax,
refs1 = refs(lc+1, a1)  a2;
refs2 = refs(lc+1, a2)  a1;
FOR r1 IN refs1 DO
IF online(peer(r1)) THEN exchange(a2, peer(r1), r+1);
FOR r2 IN refs2 DO
IF online(peer(r2)) THEN exchange(a1, peer(r2), r+1);
/* Comment:  random_select(k, refs) returns a set with k random
elements from refs
append(p1...pn, p) = p1...pn p
value(k, p1...pn) = pk
p^ - = 1+p MOD 2 */
}
```

5 Simulation

The implementation is in Mathematica[1]

There are three questions to answer:

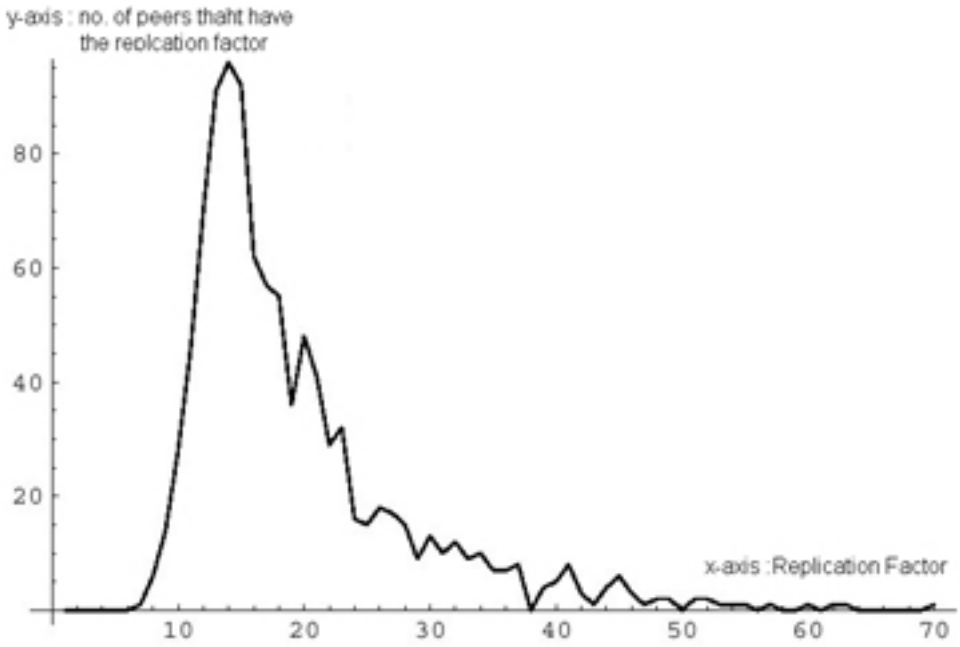
1. How many communications in terms of executing the exchange algorithm are required for building a P-Grid?

   Answer: *The results indicate that a linear relationship exists between the number of peers (N) and the total number of communications (e) needed in building the P-Grid.*

	recmax = 0		recmax=2	
N	e	$\frac{e}{N}$	e	$\frac{e}{N}$
200	15942	79.71	4937	24.68
400	27632	69.08	10383	25.95
600	43435	72.39	15228	25.38
800	59212	74.01	18580	23.22
1000	74619	74.61	25162	25.16

2. How balanced the P-Grid is with respect to the distribution of keys?

   Answer: *The average number of replication for a peer is 19.46 .*

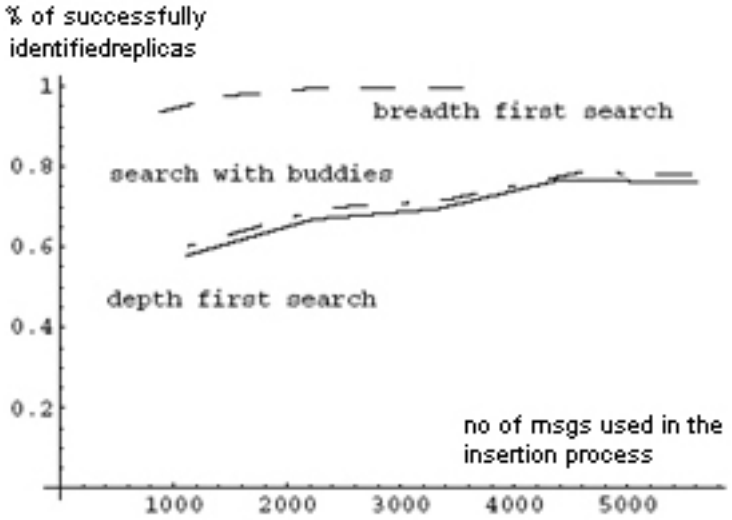


3. How reliably updates can be performed?
   Different strategies are possible:

[1] http://www.wolfram.com/

- *Randomly performing depth first searches for peers responsible for the key multiple times and propagating the update to them*
- *Performing breadth first searches for peers responsible for the key once and propagating the update to them*
- *Creating a list of buddies for each peer, i.e. other peers that share the same key, and propagate the update to all buddies.*

Using breadth first searches is by far superior:



6 Conclusions

It takes advantage of the resulting emergent properties for improving various services including routing, updates and identity management. One may also benefit from self-organizing principles when dealing with higher-level abstractions such as trust or global semantic inter operability .

What started as a purely decentralized index structure is gradually evolving into a general purpose distributed infrastructure. We have implemented P-Grid in Java and are currently in the final test phase. More information about P-Grid may be found on the project's web page at P-Grid[2]

[2]http://www.pgrid.org

Part II
Literature

Aberer01 Karl Aberer, Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. To appear in the Proceedings of the Ninth International Conference on Information and Knowledge Management (CIKM 2001) 2001.

Vingralek 98 Radek Vingralek, Yuri Breitbart, Gerhard Weikum: Snowball: Scalable Storage on Networks of Workstations with Balanced Load. Distributed and Parallel Databases 6(2): 117-156 (1998)

Stonebraker 96 Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, Andrew Yu: Mariposa: A Wide-Area Distributed Database System. VLDB Journal 5(1): 48-63 (1996)

Clarke 00 Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability. LLNCS 2009. Springer Verlag 2001.

Ratnasamy01 Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker. A Scalable Content-Addressable Network. Proceedings of the ACM SIGCOMM, 2001.