The P-Grid System - Overview

Roman Schmidt
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

March 8, 2007

1 Introduction

P-Grid [1] is an efficient infrastructure to perform lookup and search operations in large-scale and highly dynamic peer-to-peer systems. P-Grid has been designed to account for several requirements and features of P2P systems. It provides a decentralized solution where adaptability to environmental conditions is a driving factor. On the one hand, P-Grid provides an efficient structured overlay based on the concept of a distributed trie to achieve highly efficient lookup operations. Moreover, it achieves key order preservation to support range queries. On the other hand, P-Grid can utilize the underlying unstructured substrate to permit its operation in highly dynamic environments where structured approaches require high maintenance cost of the topology and the distributed index.

The basic functionality of the structured P-Grid is building a decentralized index of shared resources remaining at resource providers and performing efficient queries. The basic concepts of P-Grid will be described in more details in the following subsection, followed by an architecture overview and the implementation description.

1.1 Concepts

P-Grid is a binary trie-structured overlay network on top of the physical Internet network using prefix-based routing to provide efficient lookup operations. There are several other structured overlays which topologically resemble P-Grid and use prefix-based routing variants, for example, Pastry [7] and particularly Kademlia [6] whose XOR distance metric results in the same tree abstraction and choice of routes from all peers in complementary sub-trees as in P-Grid. The important distinguishing features of P-Grid include the emergent nature of the P-Grid network based on randomized algorithms, support for substring queries, the disentanglement of peer identifiers from the associated key space,

and the adaptive, structural replication (multi-faceted load-balancing of storage and query load) [4].

There is another motivation for having a trie-structured overlay network instead of a standard distributed hash table: The real advantage of traditionally using a hash table in main memory is the constant time of lookup, insert, and delete operations. But to facilitate this, a hash table sacrifices the order-relationship of the keys. However, over a network, where only parts of the hash table are stored at each location, we need multiple overlay hops anyway. For most conventional DHTs the number of hops is logarithmic in the network size. Thus the main advantage of constant-time access no longer exists in DHTs. This made P-Grid a natural choice for us to use it as the underlying routing network to support key search, substring search and range queries in the future, since it provides normal key search for same order of message complexity as a DHT, but in addition can be naturally extended to support range queries.

Peers construct the binary trie by pair-wise random interactions dividing gradually the key space in partitions defined by binary strings, the so-called peers' paths. For search, each peer maintains references to other peers/partitions at each level of the trie. Figure 1 shows a simple example of a P-Grid tree consisting of 6 peers responsible for 4 partitions, e.g., peer F's path is '00' leading to two entries in its routing table: peer E with path '11' at the first level and peer B with path '01' at the second level. This means that at each level of the trie the peer has references to some other peers that do not pertain to the peer's subtrie at that level which enables the implementation of prefix routing. Each peer constructs its routing table such that it holds peers with exponentially increasing distance in the key space from its own position. This technique basically builds a small-world graph [5], which enables search in $O(\log N)$ steps.

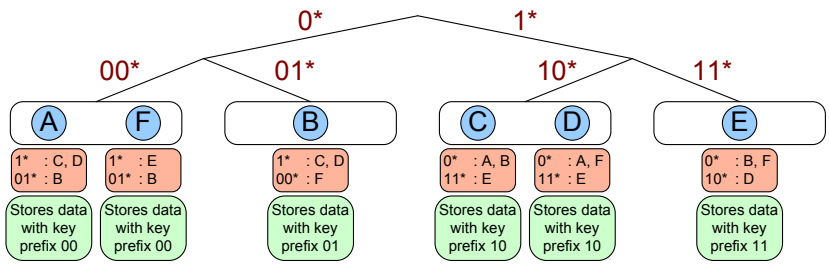

Figure 1: P-Grid overlay network

Each peer stores a set of index items which have the peers' path as prefix but it is not excluded that temporarily also other index items are stored at a peer, e.g., in Figure 1, peer F is responsible for all data with key prefix '00'. P-Grid's hash function maps application data to binary strings. In the reference implementation we assume application data to be strings for simplicity, but in fact any data type can be used. The hash function is order-preserving, i.e., it satisfies the following property for two input strings s_1 and s_2:

$$s_1 \subseteq s_2 \;\Rightarrow\; key(s_1) \subseteq key(s_2)$$

where $\subseteq$ means *is-prefix-of*.

To enable this mapping, we first constructed a balanced trie from a sample string database consisting of unique, lexicographically sorted strings of equal length (sample string databases can be provided by the user). The database is recursively bisected into equally-sized partitions until each partition is smaller than a threshold. The keys P-Grid uses are then calculated by using the application key to "navigate" character-wise through this trie and appending "0" to the generated key for each "left-turn" or "1" otherwise.

Moreover, for fault-tolerance, query load-balancing, and hot-spot handling, multiple peers are associated with the same key-space partition (structural replication), and peers additionally also maintain multiple references to peers with the same path (data replication). Both replication factors can be tailored towards domain requirements: higher structural and data replication guarantees better resilience against node failures and query-load imbalances at the cost of reducing the available capacity of the system and increasing the required system maintenance effort to keep replicas in sync and exchanging larger routing tables.

1.2 Search in P-Grid

P-Grid, like any other structured overlay approach, supports two basic operations: *Retrieve(key)* for searching a certain key and retrieving the associated index item and *Insert(key, value)* for storing new index items. Since P-Grid uses a binary tree, *Retrieve(key)* is of complexity $O(\log N)$, measured in messages required for resolving a search request, in a balanced tree, i.e., all paths associated with peers are of equal length. Skewed data distributions may imbalance the tree, so that it may seem that search cost may become non-logarithmic in the number of messages. However, in [2, 3] it is shown that due to the randomized choice of routing references from the complimentary sub-tree, the expected search cost remains logarithmic ($0.5 \log N$), independently of how the P-Grid is structured. The intuition why this works is that in search operations, keys are not resolved bit-wise but in larger blocks thus the search costs remain logarithmic in terms of messages. This is important as P-Grid uses order-preserving hashing to compute keys, which may lead to non-uniform key distributions.

The basic search algorithm is shown in Algorithm 1. p in the algorithm denotes the peer that currently processes the request.

Algorithm 1 Search in P-Grid: Retrieve(key, p)

1: **if** $\pi(p) \subseteq key$ **then**
2: return($d \in \delta(p) | key(d) = key$);
3: **else**
4: determine l such that $\pi(key, l) = \overline{\pi(p, l)}$;
5: r = randomly selected element from $\rho(p, l)$;
6: Retrieve(key, r);
7: **end if**

The algorithm always terminates successfully, if the P-Grid is complete (ensured by the construction algorithm) and at least one peer in each partition is reachable (ensured through redundant routing table entries and replication). Due to the definition of the routing table ρ and $Retrieve(key, p)$ it will always find the location of a peer at which the search can continue (use of completeness). With each invocation of $Retrieve(key, p)$ the length of the common prefix of peer p's path $\pi(p)$ and key increases at least by one and therefore the algorithm always terminates. Note that, while the network has a tree/trie abstraction, the system is not hierarchical, and all peers reside at the leaf nodes. The peer responsible for the query, i.e. the peer's path is a prefix of the key ($\pi(p) \subseteq key$), can finally answer the query by responding with all matching index entries d in the local index table $\delta(p)$.

If we consider the P-Grid tree example in Figure 1, a search initiated at peer F for key '100' would first be forwarded to peer E because it is the only entry in F's routing table at level '1*'. As peer E is responsible for '11' and not for the key '100', peer E further forwards the query to peer D, which can finally answer the query.

2 Architecture

This section will present the architecture of P-Grid implementation taking into account the gathered requirements and possible future extensions. The architecture design was driven mainly by the following requirements:

Scalability The main reason for having a distributed infrastructure instead of a centralized one is the higher scalability of decentralized solutions. The peer-to-peer architecture has to be able to support millions of users in the future sharing their knowledge and data currently only accessible at the local desktop. Whereas a centralized solution is only able support a limited number of users, scalability a decentralization is a major design requirements even though it is sometimes in conflict with other requirements for the architecture.

Fault-tolerance A decentralized system consisting of unreliable loosely coupled nodes (e.g., user desktops and laptops) has to be able to deal with failures such as network churn or node failures. The architecture has to take this into account to support those failures up to a certain degree keeping the system available without any loss of service quality.

Flexibility The P-Grid system has to be able to support future extensions by new technologies, such as security, social aspects, etc. A flexible architecture is therefore required which is able to integrate those extensions and offering new functionalities by already defined APIs. Apart from new technologies, the infrastructure also has to take into account domain specific requirements, i.e., users have to be able to tailor the P-Grid system to their needs by providing application-specific handlers.

User requirements Last but not least, user requirements received from P-Grid users have to be considered during the design of the architecture as long as they are not already covered by other requirements or can be integrated by future extensions or application-specific handlers.

The architecture of P-Grid consists of two components: (i) the P2P basic layer and the (ii) P2P index layer as shown in Figure 2. Both of them provide interfaces for applications which can either use the lower level functions of the P2P basic layer directly or the higher level functionalities of the P2P index layer on top of the basic layer.

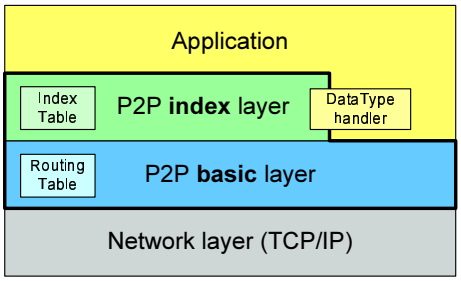

Figure 2: P-Grid architecture

P2P Basic layer The basic layer provides core functions to exploit the P2P network such as first of all joining and leaving a network. Therefore it is only necessary to know one peer of the P2P system a user wants to join. The main functionalities this basic layer provides are lookups for peers, i.e., finding a peer responsible for a given key, and routing messages to peers either given a key or already a destination peer. The lookup operation is useful for applications that wish to send messages to peers responsible for a key in a direct point-to-point manner avoiding routing the information around in the network. This is especially important for large messages that are otherwise sent via multiple hops to their destination, or confidential messages which should only be seen by the sender and the designated receiver. Messages can further be routed in the network given a key, a set of keys, and a key range. Routing a message to a single key is the basic operation of a structured overlay as described earlier to resolve a query for a known key. If the destination is defined by a set of keys or a key range, P-Grid routes the message to all peers responsible for the given keys or all peers in the given key range. It is thereby insured that all targeted peers receive exactly one message. Additionally, applications can route messages to all replicas of the local peer or simply retrieve a list of replicas and references the local peer uses to route its messages (RoutingTable).

P2P Index layer The index layer adds indexing functionalities on top of the basic layer to index and find shared content in the P2P network by reusing core functionalities from the basic layer. Applications can insert, update, and delete their content which will be hashed to the underlying key-space and routed to responsible peers which will store the new index items in their IndexTable. The application itself remains thereby responsible for which parts of the content are hashed and applications can even provide application-specific hash functions to benefit from the knowledge of the expected key distribution, improving the overall quality of the P-Grid system. This functionality has to be implemented by the DataType handlers and be provided by the application for each type of content it wishes to index and share. The DataType handler therefore enables applications and users to tailor the P2P system to their needs and optimize the infrastructure using domain-specific knowledge.

3 Implementation

The following section presents the Java implementation of the P-Grid architecture. We will mainly focus on the two P2P layers, basic and index, and neglect details about the core P-Grid implementation. More details about P-Grid can be found on the P-Grid web site[1] and in the publications available there. The two layers provide two interfaces which allow applications to use all functionalities of P-Grid and extend or tailor it to their domain-specific needs. We will therefore first introduce the two interfaces in more detail and finally describe how the DataType handler is to be used by applications.

3.1 P2P Basic Interface

The core P2P functionalities of P-Grid are accessible via the P2P basic interface. The interface itself is defined in the `p2p.basic` package whereas the P-Grid implementation of it can be found in the `pgrid.interfaces.basic` package. The offered functions are shown in Listing 1.

```
// Local operations
public void init(Properties properties);
public Peer getLocalPeer();
public Peer[] getNeighbors();
public boolean isLocalPeerResponsible(Key key);
public void shutdown();

// Join and leave functions
public void join(Peer peer);
public void leave();

// Lookup operation
public Peer lookup(Key key, long timeout);

// Routing functions
```

[1] `http://www.p-grid.org/`

```
public void route(Key key, Message message);
public void route(Key[] keys, Message[] message);
public void route(KeyRange range, Message message);
public void routeToReplicas(Message message);

// Direct Point-to-Point communication
public void send(Peer peer, Message message);

// Listener registration and removal
public void addP2PListener(P2PListener listener);
public void removeP2PListener(P2PListener listener);
```

Listing 1: The P2P basic interface

The local operations allow the application to first initialize and customize the P2P facility, e.g., by providing a listening port, before it can join the P2P network by providing a bootstrap peer, i.e., a peer which is known to the application. Further, applications might be interested in the properties their local peer has in the P2P network or which neighbors, which routing table, it uses to route messages. The lookup operation takes a key and returns a responsible peer which can be used for example by the send operation to send a message directly to a peer without routing it around in the network, e.g., with respect to the previously presented scenarios, Claudia could use the send method to directly send a task to Dirk as soon as she knows the current IP address of Dirk. The four route functions route a message to a destination key, a set of keys, a key range or simply to all replicas of the local peer, respectively. Keys are in P-Grid binary strings, e.g., '010101', and peers are identified by an unique identifier, their IP address and port as well as their path, i.e., the key partition they are responsible for. Messages can be any string or binary data applications wish to send around in the P2P network. Finally, applications can register a listener to be notified about new message arrivals with the message content and the sending peer. A notification is only created if the peer is finally responsible for a message or a direct point-to-point message was received, messages routed over a peer don't result in a notification for the application.

3.2 P2P Index Interface

The P2P index interface uses the P2P basic interface to provide higher level indexing functionality to applications. It basically allows applications to index their local content in the P-Grid network so that other users can find it efficiently. The interface itself is defined in the `p2p.index` package whereas the P-Grid implementation of it can be found in the `pgrid.interfaces.index` package. The offered functions are shown in Listing 2.

```
// Local operations
public Collection getLocalIndexEntries();
public void shutdown();

// Data modification operations
public void insert(Collection entries);
public void update(Collection entries);
```

```
public void delete(Collection entries);

// Search function
public void search(Query query, SearchListener listener)
            throws NoSuchTypeException, NoRouteToKeyException;

// Listener registration and removal
public void addIndexListener(IndexListener listener, Type type);
public void removeIndexListener(IndexListener listener, Type type);
```

Listing 2: The P2P index interface

As one of the local operations, the interface can provide a list of locally shared index entries which are currently also indexed in P-Grid. To insert, update or delete any of those items, three according functions are provided. The insert operation simply inserts the provided entry into the P-Grid network whereas the delete operation deletes all index items from the P-Grid network. The update function updates all index entries with the same index entry identifier in the P-Grid. Applications can search for content using the search function by providing a query containing either a simple keyword, multiple keywords, an upper and lower bound for range queries, etc. Results will be provided to any search listener which is registered to the query identifier, i.e., additional search listeners can be registered too. Additionally to search results, applications can receive notifications about added, removed, or updated index entries if they register as index listener.

Listing 3 shows all events a search listener can receive during an issued search operation identified by a global unique identifier (GUID). As P-Grid's communication itself is asynchronous, i.e., a query messages is sent and routed in the P-Grid network but a sending peer is not directly expecting a response, we also chose an asynchronous information flow for the implementation, realized in this case by listeners. The use of listeners has the big advantage that the search function is not blocking anymore and that more than one class implementing the search listener interface can receive search results respectively be notified about the search progress. As soon as the query message reached a responsible peer and matching items were found, the peer and therefore the listener starts to receive `newSearchResult` events. As multiple peers can be responsible for the search key space, search results may arrive sequentially leading to separate `newSearchResult` events. If no matching items could be found at a peer, the `noResultsFound` event is raised, i.e., matching items could be still found at other peers. A search is complete with the `searchFinished` notification or the `searchFailed` notification in case of network or node failures, i.e., no peers responsible could be reached.

```
public void newSearchResult(GUID guid, Collection results);
public void noResultsFound(GUID guid);
public void searchFailed(GUID guid);
public void searchFinished(GUID guid);
```

Listing 3: The SearchListener interface

3.3 DataType Handler

The data type handler enables users and applications to tailor P-Grid to their application-specific needs using prior-knowledge usable to improve the performance of P-Grid. A data type defines a type of information an application wants to share in P-Grid and is defined by an unique string, e.g., 'text/files' for simple file sharing. Applications have to create their own data types and provide a data type handler for each of them implementing the interface given in Listing 4. They therefore become responsible for core functions of P-Grid such as handling search requests, local ones as well as remote ones. The handler interface is composed of notification methods to inform the application about new or removed index items the local peer became responsible for. The idea behind letting the application be aware of those events is, that applications can extract application specific information from the index item to insert it additionally in the local database or to keep in memory. P-Grid itself is not aware of this information and therefore could not make use of it during a search. The same holds for searches, P-Grid first uses the data type handler to create query objects for a user query (`AbstractQuery[] search(pgrid.QueryInterface query)`) before it routes the queries to their responsible peer(s). There, P-Grid informs the local handler about the received query which will return the matching index items according to the query keywords. Thereby can the query contain additional information for the data type handler to further refine a query. Currently, there are search interfaces for exact keyword-based queries and range queries.

```
// modification notifications
public void indexEntryAdded(IndexEntry item);
public void indexEntryRemoved(IndexEntry item);
public void indexTableCleared();

// search handlers
public Collection handleSearch(ExactQueryInterface query);
public Collection handleSearch(RangeQueryInterface query);
public AbstractQuery[] search(pgrid.QueryInterface query);

// update handler
public boolean handleUpdate(IndexEntry item);
```

Listing 4: The DataTypeHandler interface

3.4 Usage Example

This subsection will give a short usage example of how to use P-Grid as demonstrated in Listing 5. After defining all local variables we add a new bootstrap host to the list of bootstrap hosts used by P-Grid during the bootstrap process. Those hosts are only required if a peer joins the first time a P-Grid network. After that, the basic P2P facility is acquired and initialized with the bootstrap host as property. Having a P2P basic facility, we can initialize the P2P index facility with it. No further configuration is required at this point.

To be able to share our own index entries, we have to create and register our own data type 'DemoType'. This type will be used by all index entries we will create later and by the query we will create and issue in the end. Additionally to the data type, we use the default type handler provided by P-Grid to handle our data type. The default data type handler is sufficient to share and retrieve data without making using any additional application-specific knowledge. At this point we could have provided our own data type handler to tailor P-Grid's indexing and search functionalities. Once a data type and its handler are created and registered, we are able to create index entries and inserting them in the P-Grid system. The index factory returns an index entry including an unique identifier and a binary key used to place the entry on a responsible peer.

We now join the network without the need of giving a bootstrap host as we defined one already during the initialization phase of the P2P facility. It is also possible to call the `join()` already before index entries are created but then probably additional insert messages have to be sent if a peer already joined a network and developed its own path. At the end, we create and issue a query for the keyword 'Example'. The query itself is created by the index factory given the data type we are interested in and the keyword. To receive matching items, in this example at least our two previously inserted index entries, this class also has to implement the search listener and results will be provided in the `newSearchResults(p2p.basic.GUID guid, Collection results)` function.

```
private java.util.Properties properties = new java.util.Properties();
private P2PFactory p2pFactory;
private P2P p2pService;
private IndexFactory indexFactory;
private Index indexService;
private PGridP2P pGrid = PGridP2P.sharedInstance();

// add a bootstrap peer
properties.setProperty(Properties.BOOTSTRAP_HOSTS, "myPeer.myDomain.org:1805");

// init P2P basic facility
p2pFactory = PGridP2PFactory.sharedInstance();
p2pService = p2pFactory.createP2P(properties);

// init P2P index facility
indexFactory = PGridIndexFactory.sharedInstance();
indexService = indexFactory.createIndex(p2pService);

// creating and registering data type
p2p.index.Type type = indexFactory.createType("DemoType");
TypeHandler handler = new DefaultTypeHandler(type);
indexFactory.registerTypeHandler(type, handler);

// creating some example index items
Vector items = new Vector();
IndexEntry indexItem1 = indexFactory.createIndexEntry(type, "Example 1");
IndexEntry indexItem2 = indexFactory.createIndexEntry(type, "Example 2");
items.add(indexItem1);
items.add(indexItem2);
```

```
// inserting the index entries
indexService.insert(items);

// join the P-Grid network using previously defined bootstrap hosts
pGrid.join();

Query query = indexFactory.createQuery(type, "Example");
indexService.search(query, this);
```

Listing 5: A simple usage example of P-Grid

References

[1] Karl Aberer. P-grid: A self-organizing access structure for p2p information systems. In *6th International Conference on Cooperative Information Systems (CoopIS)*, pages 179–194, London, UK, 2001. Springer-Verlag.

[2] Karl Aberer. Efficient search in unbalanced, randomized peer-to-peer search trees. Technical Report IC/2002/79, Ecole Polytechnique Fédérale de Lausanne (EPFL), 2002.

[3] Karl Aberer. Scalable data access in p2p systems using unbalanced search trees. In *4th Workshop on Distributed Data and Structures (WDAS'2002)*, 2002.

[4] Karl Aberer, Anwitaman Datta, Manfred Hauswirth, and Roman Schmidt. Indexing data-oriented overlay networks. In *31st International Conference on Very Large Databases (VLDB)*, August 2005.

[5] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. *Cornell Computer Science Technical Report*, 1776(99), 1999.

[6] P. Maymounkov and D. Maziéres. Kademlia: A Peer-to-peer Information System Based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, 2002.

[7] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.