

# Multifaceted Simultaneous Load Balancing in DHT-based P2P systems: A new game with old balls and bins\*

Karl Aberer, Anwitaman Datta, Manfred Hauswirth  
 Swiss Federal Institute of Technology Lausanne (EPFL)  
 Email: {karl.aberer, anwitaman.datta, manfred.hauswirth}@epfl.ch

## Abstract

In this paper we present and evaluate uncoordinated on-line algorithms for simultaneous storage and replication load-balancing in DHT-based peer-to-peer systems. We compare our approach with the classical *balls into bins* model, and point out the similarities but also the differences which call for new load-balancing mechanisms specifically targeted at P2P systems. Some of the peculiarities of P2P systems, which make our problem even more challenging are that both the network membership and the data indexed in the network is dynamic, there is neither global coordination nor global information to rely on, and the load-balancing mechanism ideally should not compromise the structural properties and thus the search efficiency of the DHT, while preserving the semantic information of the data (e.g., lexicographic ordering to enable range searches).

**Keywords:** Distributed Hash Tables, Uncoordinated Online Load-balancing, Storage and Replication Load, Self-organization

## 1 Introduction

Load balancing problems in P2P systems come along in many facets. In this paper we report on our results on solving simultaneously a combination of two important load balancing problems occurring in the construction and maintenance of distributed hash tables, which have conflicting requirements.

Recently distributed hash tables (DHTs) [19] have been studied intensively as a way to provide an efficient, distributed, scalable, and decentralized indexing mechanism in P2P systems. The basic principle of distributed hash tables is the association of peers with data keys and the construction of distributed routing data structures to support efficient search.

Initially, most approaches proposed to map both data and peer identifiers into a common key space, e.g., by using uniform, randomized hash functions, and in this way associating peers with the data items they are responsible to manage. The existing approaches mainly differ in the choice of topology (rings [25], multi-dimensional spaces[22], or hypercubes[24]), the specific rules for associating data keys to peer keys (closest, closest in one direction), and the strategies for constructing the routing infrastructure.

To use the available resources of peers best, a *storage load balancing* approach is applied in all DHTs, i.e., associating keys to peers in a way so that number of data items each peer is associated with is optimal in terms of storage consumption. Most existing solution use the simplistic approach of mapping keys and peer

---

\*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

identifiers into the same space and assuming a uniform distribution for both of them. Then storage load balancing essentially translates into the classical *balls into bins* problem [20], where peers are the bins (the peer identifier determines the data space) and the data items are the balls. Adapting the classical load-balancing mechanisms in the context of P2P systems, such as load-stealing and load-shedding schemes, in which peers share load with random peers, e.g., [11, 10], or power of two choices [18] (more generally, multiple hashing functions are used to determine multiple potential host peers, and least loaded one is chosen [10]), lead to the need of redirections which compromise the search efficiency, because keys become increasingly decoupled from the peers associated with the corresponding key space. This means that the search has two phases, one using the DHT's routing infrastructure, and then following the redirections, which gradually leads to deterioration of search efficiency.

The problem is further aggravated with the growing recognition of the fact that randomized hashing to generate keys which are uniformly distributed on the key space jeopardizes the possibility to do searches on data using the data key semantics, typically the ordering of keys in range searches, thus the approach of randomized key hashing for achieving uniform data distribution had to be abandoned. This implies that if peers are to be associated with keys using peer identifiers, then peer identifiers can no longer be drawn uniformly from the key space, but have to adapt to the data key distribution.

Another challenge of load-balancing in P2P system originates from the dynamic nature of P2P networks, where individual peers can autonomously (re-)join and leave the network. All approaches to mitigate the effect of network dynamics so far have used the approach to change an existing peer's identifier ([17] gives a good summary of these approaches) in order to change its associated key space, and thus trying to achieve load balance. These solutions try to solve the symptom, rather than the problem itself. Moreover, all these approaches assume a static distribution of data, and lack adaptability to changes in the data distribution without compromising the structural properties of the DHT, and hence the search efficiency.

Another possible approach, which we will follow in this paper, is to have peers dynamically change their associated key space ("bin adaptation") decoupled from their identifier, and the routing between peers is based on the associated key space, rather than on the peer identifiers. Following this approach, the partitioning of the data space dynamically adapts to any granularity, until uniform distribution of data items over each partition of the key space is achieved. This leads to uneven sizes of the partitions of the key space, which can be viewed analogously to having an unbalanced search tree. This implies a risk of sacrificing search efficiency. However, we show that due to the distributed and randomized search process we use (P-Grid), this risk can be contained, such that searches can be performed with communication cost of  $O(\log(|\Pi|))$  with high probability where  $|\Pi|$  is the number of partitions of the key space, irrespective of the key space partitioning. This satisfies the condition of efficient searches in the context of P2P systems under the (standard) assumption that in a P2P network, local resources such as computation and storage are cheap, but communication costs (messages) and network maintenance (routing) are expensive.

Beyond search efficiency, another important issue in P2P systems is resilience against failures. The standard response to this problem is to introduce redundancy. In the context of DHTs this corresponds to associating multiple peers with the same partition of the search space, i.e., peers being replicas of each other. A fair use of resources implies uniform replication of all data partitions, which we call the *replication load balancing* problem. The initial approach to balance replication was to have a predefined constant number of replicas for each data partition [25, 22, 23, 8]. However, the difficulty is to determine the correct replication constant in absence of global knowledge (e.g., total peer population size, total storage space, total data load) and the lack of adaptivity to distribute the resources in a dynamic environment (change in the peer membership, or the data in the network) in a fair way.

An alternative approach is to determine the replicas for each data space partition dynamically which again induces a load balancing problem, i.e., how to assure that each partition is associated with approximately the same number of replica peers. Here, the key space partitions are the bins, and the peers are the balls. This problem could again be solved by standard distributed load balancing algorithms if the data partitions were

known. However, as mentioned before, determining data partitions by itself is a different load balancing problem to solve.

In this paper we use a DHT (called P-Grid) which decouples the peer identifier from the associated key space and routing, and hence has greater flexibility for balancing load (both storage and replication). We have proven in past work that because of the randomization in the choice of routing options in P-Grid, search efficiency is retained for any arbitrary key distribution [2], i.e., it is possible to have a data structure that meets this requirements.

In this paper we provide a decentralized construction algorithm for building such a distributed data structure which adapts the key space partitioning to the data distribution to ensure storage load balancing. The construction mechanism also has an element of maintenance capability inherent to it, such that, if over time the data distribution changes, the key space partitioning will change as well (by mergers and repartitioning of key spaces). The construction process, however, does not ensure replication balancing, and hence we propose a maintenance algorithm which decreases the variance of replication factors. Note that even if the construction algorithm were to create a P-Grid with perfect replication balancing, we still needed the maintenance mechanism in order to cope with changing membership, for instance, if peers leave the network, rendering a key partition overloaded. Since global coordination and knowledge cannot be assumed in a decentralized environment, the maintenance algorithm relies on each peer obtaining an approximate local view of the system based on sampling (random interaction with some other peers, for instance while answering queries) and making an autonomous probabilistic decision to replicate an overloaded key space prioritized according to the load-imbalance between two sub-partitions of the key space.

Our approach has several advantages: (a) By disentangling the peer identifier from the associated key space, we not only have greater flexibility for load balancing such that the system can adapt to changing data distributions and network membership changes, but it also preserves peer identity (unlike other proposed approaches for storage load balancing in DHTs [17]). Preservation of peer identity may not be critical for the existing file sharing networks, but is important for information systems applications that uses a P2P network as the underlying infrastructure. (b) We address multifaceted load-balancing concerns simultaneously in a self-organizing manner without assuming global knowledge, or restricting the replication to a predetermined number. (c) We preserve key ordering which is important for range queries, while retaining the search efficiency  $O(\log(|\Pi|))$  in terms of the number of key space partitions  $|\Pi|$ , irrespective of the key distribution. (d) We retain the structural properties of the distributed hash table, and do not compromise the search efficiency in presence of storage balancing unlike other approaches mentioned above. The presented approach is implemented in our P-Grid system [7] which is available at <http://www.p-grid.org/>.

## 2 The P-Grid data structure

We use our DHT-based P-Grid P2P system [1, 4] to evaluate the approach described in this paper. We assume that the reader is relatively familiar with the standard distributed hash table (DHT) approach [19] and thus only provide P-Grid's distinguishing characteristics. We focus on the essential P-Grid concepts that are required to understand our approach and introduce the necessary notations. Then we will prove that, unlike other existing DHTs, the P-Grid structure can accommodate arbitrary key distributions without compromising either storage load balancing or search efficiency, such that, irrespective of the key distribution, searches are logarithmic in terms of key space partitioning, where the key space is partitioned unevenly in order to accommodate storage load balance.

In P-Grid, peers refer to a common underlying tree structure in order to organize their routing tables (as opposed to other topologies, such as rings [25], multi-dimensional spaces[22], or hypercubes[24]). In the following, for simplicity of presentation 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 [5]. 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 [2].

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  $\pi(p, l) = \pi(q, l)$ , where  $\bar{\pi}$  is the binary string  $\pi$  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. 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 other data items are also 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  $\pi(q) = \pi(p)$ . 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.

## 2.1 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  $\kappa$ . To search for a key  $\kappa$  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  $\kappa$ . If yes, it checks whether it can return a query result from its data store. If not, it selects a peer  $r$  having a common prefix of maximal length with  $\kappa$  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  $\kappa$  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. Skewed data distributions may imbalance the tree, so that it may seem that search cost may become non-logarithmic in the number of messages needed to locate the peer holding a searched data item. However, in [2] 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.

**Theorem 1.** *The expected search cost for the search of a specific key  $\kappa(d)$  using a P-Grid network  $N$  that is randomly selected among all possible P-Grids, starting at a randomly selected peer  $p$  with  $\pi(p) \in \Pi$  is less than  $\log(|\Pi|)$ .*

Although this applies to the special case of prefix-free P-Grids, we have shown by simulation that the result also applies to more general cases. A formal proof of this theorem is given in [2]. Due to space limitations we can only provide the intuition which is underlying the proof. Basically we show that the path resolution in the forwarding process normally is not done bit by bit but for longer bit sequences at the processing peers thus keeping the number of messages required in the forwarding process logarithmic. Additionally, [2] shows that the probability that a search does not succeed after  $k$  steps ( $1 \leq k \leq \max(|\pi|, \pi \in \Pi)$ ) is smaller than  $\frac{\log(n)^{k-1}}{(k-1)!}$ .

### 3 A motivating scenario

In the following we provide an elementary example to illustrate constructing and maintaining a P-Grid. We assume 6 peers, each of them being able to store two data items. Let us assume that initially 6 data items are stored by the peers. The states of the peers are represented by triples  $[a, \pi(a), \delta(a)]$ . Each peer stores some data and initially all paths are empty ( $\epsilon$ ), i.e., no P-Grid has been built yet. For the data items we assume that their corresponding keys have the following 2-bit prefixes:  $\kappa(A_i, 2) = 00$ ,  $\kappa(B_i, 2) = 10$ ,  $\kappa(C_i, 2) = 11$  ( $i = 1, 2$ ).

Action	Resulting state.
Initial state.	$[P_1, \epsilon, \{A_1, B_1\}] [P_2, \epsilon, \{B_1, C_1\}]$ $[P_3, \epsilon, \{A_2, B_2\}] [P_4, \epsilon, \{B_2, C_2\}]$ $[P_5, \epsilon, \{A_1, C_1\}] [P_6, \epsilon, \{A_2, C_2\}]$
$P_1$ initiates a P-Grid network $N_1$ . $P_2$ joins the network by contacting $P_1$ . We assume that whenever at least 1 data item pertaining to a subspace is available a peer attempts to specialize to that subspace. Thus $P_1$ and $P_2$ can split the search space.	$N_1 : [P_1, 0, \{A_1\}], [P_2, 1, \{B_1, C_1\}]$
Independently $P_3$ starts a P-Grid network $N_2$ and $P_4$ joins this network.	$N_2 : [P_3, 0, \{A_2\}], [P_4, 1, \{B_2, C_2\}]$
Next $P_5$ joins network $N_1$ by contacting $P_2$ . Since $\pi(P_2) = 1$ , $P_5$ decides to take path 0.	$N_1 : [P_1, 0, \{A_1\}], [P_2, 1, \{B_1, C_1\}],$ $[P_5, 0, \{A_1\}]$
Now $P_6$ enters network $N_1$ by contacting $P_5$ . Since $\pi(P_5) = 0$ , $P_6$ decides to adopt 1 as its path and sends $\{d \in \delta(P_6)   \kappa(d) = 0\} = \{A_2\}$ to $P_5$ which stores it.	$N_1 : [P_1, 0, \{A_1\}], [P_2, 1, \{B_1, C_1\}],$ $[P_5, 0, \{A_1, A_2\}], [P_6, 1, \{C_2\}]$
Next $P_3$ contacts $P_1$ . As a result the two networks $N_1$ and $N_2$ merge into a common network $N$ and become a single P-Grid. This shows that P-Grids do not require to start from a single origin, as assumed by standard DHT approaches, but can dynamically merge, similarly to unstructured networks. Since $\pi(P_3) = \pi(P_1) = 0$ and they still have extra storage space, they can replicate their data to increase data availability.	$N : [P_1, 0, \{A_1, A_2\}],$ $[P_2, 1, \{B_1, C_1\}],$ $[P_3, 0, \{A_1, A_2\}],$ $[P_4, 1, \{B_2, C_2\}],$ $[P_5, 0, \{A_1, A_2\}],$ $[P_6, 1, \{C_2\}]$
In order to explore the network $P_2$ contacts $P_4$ . Network exploration serves the purpose of network maintenance and can be compared to the ping/pong protocol used in Gnutella. $\pi(P_2) = \pi(P_4) = 1$ they can now further refine the search space by specializing their paths and exchange their data according to the new paths.	$N : [P_1, 0, \{A_1, A_2\}],$ $[P_2, 10, \{B_1, B_2\}],$ $[P_3, 0, \{A_1, A_2\}],$ $[P_4, 11, \{C_1, C_2\}],$ $[P_5, 0, \{A_1, A_2\}],$ $[P_6, 1, \{C_2\}]$
Apparently all peers except $P_6$ have now specialized to the maximum possible degree. So what will happen to $P_6$ ? It may eventually contact first $P_2$ and decide to specialize to $\pi(P_2) = 11$ and later encounter $P_4$ and obtain the missing data item pertaining to path 11. This is the final state.	$N : [P_1, 0, \{A_1, A_2\}],$ $[P_2, 10, \{B_1, B_2\}],$ $[P_3, 0, \{A_1, A_2\}],$ $[P_4, 11, \{C_1, C_2\}],$ $[P_5, 0, \{A_1, A_2\}],$ $[P_6, 11, \{C_1, C_2\}]$

The resulting P-Grid is now not only complete, but also prefix-free. The storage load for all peers is perfectly balanced, as a result of the local decisions made to exchange and replicate data and specialize

paths. Globally, however, the replication factors are not balanced. There exist three peers that support path 0, two that support path 11, and only one supports path 10. Consequently data items pertaining to path 0 are replicated more often and thus better available. This imbalance resulted from the specific sequence of interactions performed. Other sequences would have led to other, possibly more balanced replication. However, since no global coordination can be assumed, we cannot exclude such “undesired” sequences of events.

In the following, we will introduce randomized algorithms requiring no central coordination that will reduce global imbalance of replication factors and at the same time maintain local storage balance during construction of P-Grids. Moreover, in case such imbalances occur as a result of the construction or due to changing data distributions, they will re-balance the structure. In our example such re-balancing could be achieved if one of the peers supporting path 0 decided to replicate path 10 instead. The difficulty for the algorithms lies in the fact, when and how to decide on such changes to the P-Grid structure if they can base their decision only on locally available information. The heuristics for taking these decisions have to be chosen very carefully so that the overall load-balancing goal is supported and not hampered mistakenly.

## 4 P-Grid construction algorithm

The construction and maintenance of P-Grid is based exclusively on local interactions among peers in order to observe the principle of locality. In this section we give an overview of the *possible* interactions. They determine the behavioral options of peers. As peers are autonomous they may use different *strategies* for entering into such local interactions. The choice of *concrete* strategies will be essential with respect to the global efficiency of the system and discussed later.

For the purpose of keeping the presentation clear and modular we will specify the local interactions by means of event-condition-action (ECA) rules, which are a well established concept from active databases. Events will relate to the occurrence of communication among peers, conditions will apply to the states of the interacting peers, and actions correspond to restructuring of the peers’ states and to the generation of further interactions, i.e., events.

The only event we consider for maintenance is *invitation*, i.e., peer  $p$  invites peer  $q$  to perform a maintenance interaction, which we denote as  $p \rightarrow q$ . We will now discuss informally the possible interactions and their purpose for the invitation event which can be autonomously initiated by any peer. Note that these rules only provide *necessary* conditions for their application. Later we will enhance them with further conditions which will be considered as being *sufficient* in order to implement *strategies*, that lead to certain desired global behaviors emerging from the aggregate local interactions.

- *balSplit*( $p, q$ ): The peers check whether their paths are identical. If yes, they extend their paths by complementary bits, i.e., split the common subspace covered by their paths, exchange their data correspondingly and reference each other for future query routing. This enables the refinement of the indexing structure into subspaces which are sufficiently populated with data. In the following we will interchangeably refer to “path extensions with complimentary bits” and “key space partitioning.”
- *unbalSplitL*( $p, q$ ): The peers check whether  $\pi(p)$  is a proper prefix of  $\pi(q)$ . If yes,  $p$  extends its path by one bit complementary to the bit of  $\pi(q)$  at the same level. The peers exchange their data corresponding to the updated paths and update their routing table. This enables the refinement of the indexing structure into subspaces as in the previous case, but covers the frequently occurring situation that peers have already specialized to different degrees. *unbalSplitR*( $p, q$ ) is defined accordingly if  $\pi(q)$  is a proper prefix of  $\pi(p)$ .
- *balDataExchange*( $p, q$ ): The peers check whether their paths are identical. They replicate mutually

all data pertaining to their common path. This allows provides the possibility to take advantage of unused storage space to increase resilience.

- *unbalDataExchangeL(p, q)*: The peers check whether  $\pi(p)$  is a proper prefix of  $\pi(q)$ . If this is the case, data of  $p$  pertaining to  $\pi(q)$  is moved to  $q$ . *unbalDataExchangeR(p, q)* is defined accordingly if  $\pi(q)$  is a proper prefix of  $\pi(p)$ .
- *refExchange(p, q)*: The peers exchange randomly entries from their routing tables up to the level corresponding to the length of their common prefix. This interaction randomizes the contents of the routing tables which is essential to maintain routing efficiency, in particular in unbalanced P-Grids [2].
- *forwarding(p, q)*: The peer  $q$  receiving the invitation provides the inviting peer  $p$  with an address of a peer  $r$  selected from its routing table which shares a prefix of maximal length with  $\pi(p)$ . Then peer  $p$  issues an invitation event  $p \rightarrow r$ , i.e., enters into an interaction with peer  $r$ .

For illustration we provide formal specification of two sample ECA rules. We use the following path related notations:  $\pi_1 \cap \pi_2$  is the common prefix of  $\pi_1$  and  $\pi_2$ ,  $\pi + b$  is the path  $\pi$  extended by bit  $b$ , and  $|\pi|$  is the length of a path.  $\gamma(S)$  is a uniformly randomly element from set  $S$  and  $\gamma(S, l)$  is a set of  $l$  uniformly randomly selected elements from  $S$ . Frequently actions are applied symmetrically to both peers. In those cases the notation  $p||q$  is used to express that the action is executed with both the first and the second component instantiated.

$$\begin{aligned}
 \textit{balSplit}(p, q) & : \\
 & \textbf{on } p \rightarrow q \\
 & \textbf{if } \pi(p) = \pi(q) \\
 & \textbf{then } b := \gamma(\{0, 1\}); \pi(p||q) := \pi(p||q) + b||\bar{b}; \\
 & \quad \rho(p||q, |\pi(p||q)|) := \{q||p\}; \\
 & \quad \delta(p||q) := (\delta(p||q) \setminus \delta(p||q, \pi(q||p))) \cup \delta(q||p, \pi(p||q)); \\
 \textit{refExchange}(p, q) & : \\
 & \textbf{on } p \rightarrow q \\
 & \textbf{if } \textbf{true} \\
 & \textbf{then } \forall l \leq |\pi(p) \cap \pi(q)| \rho(p||q, l) := \gamma(\rho(p, l) \cup \rho(q, l), \rho_{max}); \\
 & \quad \textbf{if } |\pi(p||q)| > |\pi(p) \cap \pi(q)| \textbf{then} \\
 & \quad \quad \rho(p, |\pi(p||q) \cap \pi(q||p)| + 1) \\
 & \quad \quad := \sigma(\rho(p||q, |\pi(p||q) \cap \pi(q||p)| + 1), \rho_{max} - 1) \cup \{q||p\}
 \end{aligned}$$

From elementary rules, strategies can be defined by associating sequences of rules with peers and refining the rule conditions. We denote a refinement of a rule *rule* : **on event if condition then action** by a condition *refcond* as *rule / refcond*. The semantics of the refined rule is given by the modified rule *rule / refcond* : **on event if condition  $\wedge$  refcond then action**.

The following sequence of rules provides a possible strategy to construct a P-Grid structure from an initial state (all peers store some initial data and have empty paths and routing tables). Note that the rules are evaluated by the peers in the given order and all peers share the same rule set.

$$\textit{refExchange}(p, q);$$

$$\begin{aligned}
 & \text{balDataExchange}(p, q) / |\delta(p, \pi(p)) \cup \delta(q, \pi(q))| \leq 2\delta_{max}; \\
 & \text{balSplit}(p, q) / |\delta(p, \pi(p)) \cup \delta(q, \pi(q))| > 2\delta_{max}; \\
 & \text{unbalSplitL}(p, q); \\
 & \text{unbalSplitR}(p, q); \\
 & \text{forwarding}(p, q);
 \end{aligned}$$

Note that due to the *forwarding* rule any initial invitation will eventually lead to the enabling of one of the balanced or unbalanced split or data exchange operations.

This strategy will eventually converge to a state where each peer carries at most  $2\delta_{max}$  data items and all peers with equal paths have exactly the same data items ( $\delta_{max}$  denotes the number of data items a peer wants to store on average and  $2\delta_{max}$  is the maximum number of data items it is willing to store in the long run).

**Theorem 2.** *The P-Grid construction algorithm results in a steady state in which the P-Grid is prefix-free and complete and each peer  $p$  with replica has a data load smaller than  $2\delta_{max}$  and all replicas store the same data.*

*Proof Sketch:* First we have to show that the steady state is reached. Prefix-freeness follows from the fact that whenever a peer has a path that is a prefix of the path of another peer, it eventually will encounter this peer and perform an unbalanced split. Completeness follows from the fact that new paths can only occur as the result of a balanced split. If a peer has a replica and the data load is larger than  $2\delta_{max}$ , it will eventually perform a split with its replica. If peers with the same path have different data items then they will eventually perform a balanced data exchange.

Second, it is easy to see that once the steady state is reached none of the rules can induce further changes to the paths or data associated with the peers.  $\square$

The problem is that with this strategy peers preferably adapt shorter paths and therefore even though peers try to balance their storage load, the distribution of replicas over the different paths becomes unbalanced. This is easy to see:

- In a balanced split the same number of peers decide for each side of the data space independent of the actual distribution of data among the two subspaces.
- In an unbalanced split peers decide for one side with a probability proportional to the number of peers already specialized for each side of the data space, but independent of the number of data items present in the two subspaces.

This has the further effect that fewer peers specialize on paths with higher data load, end sooner end up without replicas and thus lack the capacity to further refine the path and thus reduce their data load.

Therefore we modified the strategy to achieve an improved replica balancing already during construction of the P-Grid structure.

$$\begin{aligned}
 & \text{refExchange}(p, q); \\
 & \text{balDataExchange}(p, q) / |\delta(p, \pi(p)) \cup \delta(q, \pi(q))| \leq 2\delta_{max}; \\
 & \text{balSplit}(p, q) / |\delta(p, \pi(p)) \cup \delta(q, \pi(q))| > 2\delta_{max} \wedge \gamma([0, 1]) < \alpha; \\
 & \text{unbalSplitL}(p, q) / |\delta(p, \overline{\pi(q)}, |\overline{\pi(p)}| + 1)| > \delta_{max}; \\
 & \text{unbalSplitR}(p, q) / |\delta(q, \overline{\pi(p)}, |\overline{\pi(q)}| + 1)| > \delta_{max}; \\
 & \text{unbalDataExchangeL}(p, q) / |\delta(p, \overline{\pi(q)}, |\overline{\pi(p)}| + 1)| \leq \delta_{max}; \\
 & \text{unbalDataExchangeR}(p, q) / |\delta(q, \overline{\pi(p)}, |\overline{\pi(q)}| + 1)| \leq \delta_{max}; \\
 & \text{forwarding}(p, q);
 \end{aligned}$$

In this strategy two mechanisms work together to improve replica balancing. First, balanced splits are not performed eagerly, but with reduced probability  $\alpha$ . As a result more unbalanced split situations occur. In those situations peers only extend their path if they encounter data exceeding the maximal storage load  $2\delta_{max}$  in the corresponding peers subspace. As a result those subspaces will be populated by more peers that contain more data. Even though, this heuristic approach too does not immediately induce a perfectly uniform replica distribution, it substantially improves the state reached after the P-Grid construction. The remaining balancing is then achieved by the sampling-based maintenance algorithms, that we will introduce subsequently. Having a more uniform initial replica distribution, substantially reduces the effort required from the maintenance algorithms in order to rectify the distribution.

The construction algorithm can be extended to a maintenance algorithm. If data distribution changes, it may be necessary to retract paths (also referred to as key subspace merger) as well as use the already explained path extensions, in order to restructure the P-Grid tree to the new steady state conforming to the latest key distribution. The path retraction is dual to the path extension, such that if two partitions do not have enough data ( $< \delta_{max}/2$ ), then such partitions would be merged. Further details of the restructuring algorithm have been omitted because of space constraints, but we do provide results of the restructuring process.

## 5 Maintenance algorithm

As described in the previous section, storage load balancing by means of adaptation of the tree structure may lead to non-uniform replica distribution. Also dynamic off-online behavior of peers may compromise replica balancing. Thus an algorithm for re-establishing uniform replication of data is required, which amounts to solving a global coordination problem.

We use a reactive randomized distributed algorithm which tries to achieve globally uniform replication based on locally available information. Before introducing the algorithm we introduce the principles underlying its design.

### 5.1 Think local, act global

Consider the case of a P-Grid with two leaves, as shown in Figure 1(a). Let  $N_1 > N_2$  be the actual number of replica peers with paths 0 and 1. To achieve perfect balancing  $\frac{N_1-N_2}{2}$  of the peers with path 0 would need to change their path to 1. Since each of the peers has to make an autonomous decision as to whether to change its path, we propose a randomized decision, such that peers decide to change their paths with probability  $p_{0 \rightarrow 1} = \max(\frac{N_1-N_2}{2N_1}, 0)$ . The function ensures that no  $0 \rightarrow 1$  transition occurs if  $N_2 > N_1$ .

Now, if we set  $p_0 = \frac{N_1}{N_1+N_2}$  as the probability that peers have path 0, and similarly  $p_1 = \frac{N_2}{N_1+N_2}$ , then we can rewrite the migration probability as  $p_{0 \rightarrow 1} = \max(\frac{1}{2}(1 - \frac{p_1}{p_0}), 0)$ . It is easy to see that with this transition probability on average an equal replication factor is achieved for each of the two paths after each peer has taken the migration decision. In a practical setting peers do not know  $N_1$  and  $N_2$ , but they can easily determinate an approximation of the ratio  $\frac{N_1}{N_2}$  by keeping statistics of which types of peers they encounter during random interactions initiated for maintenance purposes.

Now consider the case of a P-Grid with three leaves, as shown in Figure 1(b), with  $N_1$ ,  $N_2$  and  $N_3$  replicas for the paths starting with 0, 10 and 11 respectively. This extension of the example captures the essential choices that have to be made by individual peers in a realistic P-Grid, and provides an insight for the design of the replication balancing strategy.

In an unbalanced tree, knowing the count of peers for the two sides at any level is not sufficient because, even if replication is uniform, the count will provide biased information, with a higher value for the side of the tree with more leaves. Such count in itself is useless. On the other hand, knowledge of the whole tree (shape and replication) at all peers is not practical. Fortunately, such knowledge is not necessary either.

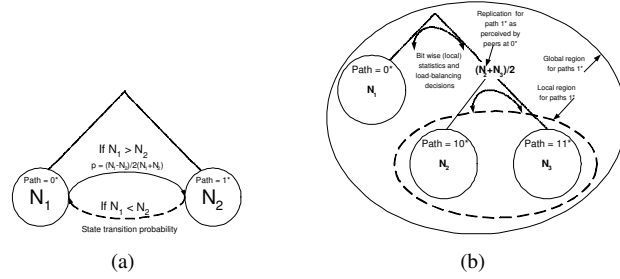


Figure 1: (a) P-Grid with two leaves, (b) P-Grid with three leaves

For example, in the P-Grid with three leaves, peers with path 0 will meet peers with paths 10 and 11. Essentially, they need to know that there are on an average  $\frac{N_2 + N_3}{2}$  peers at each leaf of the other sub-tree, but do not need to understand the shape of the sub-tree or the distribution of replication factors.

Thus, while collecting the statistical information, any peer  $p$  counts the number of peers encountered with common prefix length  $l$  for all  $0 \leq l \leq |\pi(p)|$ . It normalizes the count through dividing it by  $2^{|\pi(q)| - |\pi(p) \cap \pi(q)|}$ . Thus peers obtain from local information an approximation of the global distribution of peers *pertaining to their own path*. The latter aspect is important to maintain scalability.

In our example peers with path 0 will count on average  $\frac{N_2 + N_3}{N_1}$  as many occurrences of peers with path 10 or 11 than they will count with path 0, but will normalize their count by a factor of  $\frac{1}{2}$ . Thus at the top level they will observe replica balance exactly if on average  $N_1 = \frac{1}{2}(N_2 + N_3)$ . If imbalance exists they will migrate with probability  $p_{0 \rightarrow 1} = \max(\frac{1}{2}(1 - \frac{p_1}{p_0}), 0)$ , where, now  $p_0 = \frac{N_1}{N_1 + \frac{1}{2}(N_2 + N_3)}$  and  $p_1 = \frac{\frac{1}{2}(N_2 + N_3)}{N_1 + \frac{1}{2}(N_2 + N_3)}$ .

Once balance is achieved at the top level, peers at the second level with paths 10 and 11 will achieve balance as described in the first example. Thus local balancing propagates down the tree hierarchy till global balance is achieved. The peers with longer paths may have multiple migration choices, such that balancing is performed at multiple levels simultaneously. For example, if  $N_1 = N_2 < N_3$  peers with path 11 can choose migrations  $11 \rightarrow 0$  and  $11 \rightarrow 10$  with equal probability.

Note that  $N_i$  changes over time, and thus the statistics have to be refreshed and built from scratch regularly. We can thus consider the algorithm to have two phases, one that gathers statistics, and the other one that makes the probabilistic decision to migrate. These ideas are the basis for the statistics collection and migration strategy which we will describe later.

To demonstrate the effectiveness of the idea we performed a numerical simulation of the approach for the P-Grid with three leaves. Starting from different initial combinations of  $N_1, N_2, N_3$ , in each round in each time step  $t$ , each peer determines the transition probability based on a complete statistics of the current peer population and then performs its decision based on these probabilities. The resulting expected replication vectors are shown in Table 1. We can see that the strategy is robust, irrespective of the initial replication distribution.

$t$	$N_1(t)$	$N_2(t)$	$N_3(t)$	$N_1(t)$	$N_2(t)$	$N_3(t)$	$N_1(t)$	$N_2(t)$	$N_3(t)$
0	15	20	30	20	30	15	30	20	15
1	21	24	20	21.25	21.25	22.5	23.75	20.625	20.625
2	21.5	21.5	22				22.2	21.4	21.4

Table 1: Number of replicas over time

## 5.2 Balancing replication load

We now generalize the principles of the previous section to concrete algorithms for maintaining replication balancing in a P-Grid. Each peer determines the replication factors for all possible prefixes of its path and those of the complementary subspaces, by collecting a minimal number of samples for each level. The sampling can be done in a proactive manner specifically for the purpose of load-balance maintenance, or during random interaction of peers that occur when they communicate during query forwarding or route maintenance operations.

**Collecting statistical information at peer  $p$ :** In a decentralized setting, a peer  $p$  has to rely on sampling to obtain an estimate of the global load imbalance. In order to do so, upon meeting any random peer  $q$ , peer  $p$  will gather statistical information for all possible levels  $l \leq |\pi(p)|$  of its path, and update the number of peers belonging to the same subspace  $\Sigma_p(l) = |\{q \text{ s.t. } |\pi(p) \cap \pi(q)| \geq l\}|$  and the complimentary subspace  $\overline{\Sigma}_p(l) = |\{q \text{ s.t. } \overline{\pi(p, l)} = \pi(q, l)\}|$  at any level  $l$ . Gathering statistics is captured by the following rule:

**on**      $p \rightarrow q$   
**then**     $l := |\pi(p) \cap \pi(q)|;$   
            $\overline{\Sigma}_{p||q}(l) := \overline{\Sigma}_{p||q}(l) + 2^{1+l-|\pi(q||p)|};$   
            $\forall 0 \leq i < l \ \Sigma_{p||q}(i) := \Sigma_{p||q}(i) + 2^{1+i-|\pi(q||p)|};$

**Choosing migration path for peer  $p$ :** A path change of a peer only makes sense if this does incur a reduction of the number of replicas in a subspace that is already underpopulated. Therefore, as soon as a minimum number of samples has been obtained, the peer tries to identify possibilities for migration. It determines the largest  $l_{max}$  such that

$$\frac{\Sigma_p(l_{max})}{\overline{\Sigma}_p(l_{max})} > \zeta$$

where  $\zeta \geq 1$  is a dampening factor which avoids migration if load-imbalance is within a  $\zeta$  factor. We set  $l_{max} := \infty$  if no level satisfies the condition.

If all peers try to migrate to the least replicated subspace, we would induce an oscillatory behavior such that the subspaces with low replication would turn into highly replicated subspaces and vice versa. Consequently, instead of greedily balancing load, peers essentially have to make a probabilistic choice proportional to the relative imbalance between subspaces. Thus  $l_{migration}$  is chosen between  $l_{max}$  and  $|\pi(p)|$  with a probability distribution proportional to the replication load-imbalance  $\frac{\Sigma_p(i)}{\overline{\Sigma}_p(i)}, |\pi(p)| \geq i \geq l$ . Thus the migrations are prioritized to the least populated subspace from the peer's current view, yet ensuring that the effect of the migrations is fair, and not all take place to the same subspace. There are subtle differences in our approach to replication balancing in comparison to the classical balls into bins load balancing approach, because in our case there are no physical *bins*, which would share load among themselves, and it is rather the *balls* themselves, which need to make an autonomous decision to migrate. Moreover, the load sharing is not among bins chosen uniformly, but is prioritized based on locally gathered approximate global imbalance knowledge.

To further reduce oscillatory behavior, the probability of migration is reduced by a factor  $\xi \leq 1$ . As migration is an expensive operation—it leads to increased network maintenance cost due to routing table repairs [3]—it should only occur if long-term changes in data and replication distribution are observed and not result from short term variations or inaccurate statistics. The parameters  $\zeta$  and  $\xi$  are design parameters and the impact of their choice on the system behavior will be further explored in Section 6.

**Migrating peer  $p$ :** The last aspect of replication load balancing is the action of changing the path. For that, peer  $p$  needs to find a peer from the complimentary subspace. To identify such a peer, it inspects its routing table  $\rho(p, l_{migration})$  (s.t.  $\pi(p) \cap \pi(q) = l_{migration}$ ). After identifying a peer  $q$ ,  $p$  clones the contents of  $q$ , including data and routing table, i.e.,  $\delta(p) := \delta(q)$  and  $\rho(p, *) = \rho(q, *)$ , and the statistical information is reset in order to account for the changes in distribution.

## 6 Simulation results

This section presents a brief summary of the numerous experiments we performed using a simulator implemented in Mathematica 4.2 in order to evaluate the proposed construction and maintenance algorithms. The simulations aim at verifying the load balancing characteristics of the algorithms, and do not model aspects related to the physical runtime environment with different network topologies, communication latencies, or heterogeneity of resources of nodes.

Unless mentioned otherwise, simulations were performed with 256 peers. This relatively low number was chosen to keep simulation time manageable. From the design of the algorithms it is clear that the results will scale up to larger populations. To support this, we will give one result for the complete maintenance algorithm with changing peer population at the end. The data was chosen from a Zipf distribution with parameter  $\theta = 0.8614$  such that the frequencies of keys were monotonically increasing with decreasing size of the key. We set  $\delta_{max} = 50$ .

**Replication load balancing throughout construction.** In Section 4 we discussed a possibility to maintain better replica load balancing while establishing storage load balance during P-Grid construction, by reducing the probability  $\alpha$  of balanced splits of the key space. In Table 2 we show the results of an experiment in which each peer initially holds 15 data items. During P-Grid construction we see how a reduction of  $\alpha$  reduces both the variance  $R_{\sigma^2}$  in replication factor (i.e., number of peers with the same path) and the maximum replication factor  $R_{max}$ , where  $R_{\mu}$  is the average replication factor with an expected value of  $3.33^1$ . With lower probabilities more exchange operations had to occur in order to reach a steady state.

$\alpha$	Exchanges	$R_{\mu}$	$R_{\sigma^2}$	$R_{max}$
0.05	40,000	3.32	1.82	10
0.1	35,000	3.20	1.99	9
0.5	20,000	3.55	3.39	21
1.0	20,000	3.28	3.94	23

Table 2: Influence of splitting probability  $\alpha$  on replication and path variance

**Replication load balancing throughout maintenance.** Given a P-Grid that partitions the data space such that the storage load is uniform for all partitions, migrations are used to establish simultaneous balancing of replication factors for the different partitions without changing the data space partitioning.

For the experiments we chose the design parameters  $\zeta = 1.1$  (required imbalance for migration),  $\xi = 0.25$  (attenuation of migration probability) and a statistical sample size of 10. These parameters had been determined in initial experiments as providing stable (non-oscillatory) behavior.

The performance of the migration mechanism depends on the number of key space partitions and the initial number of peers associated with each partition. Since the expected depth of the tree structure grows

<sup>1</sup>There are  $256 * 15$  data items, on average 50 of them are stored at each peer. They require  $\frac{256 * 15}{50}$  partitions to be replicated among 256 peers, which results in  $\frac{50}{15} = 3.33$  replicas on average.

logarithmically in the number of partitions, and the maintenance is expected to grow linearly with the depth of the tree (since each peer uses its local view for each level of its current path), we expect the maintenance algorithm to have logarithmic dependency between the number of partitions and the rate of convergence.

In Figure 2(a) we show the reduction of replication factor variance compared with the initial variance as a function of the number of partitions. The results were obtained for replication balance using peer migration, but no P-Grid restructuring, starting from initial, unbalanced key space partitions, with the number of partitions being  $p = \{10, 20, 40, 80\}$ , and each partition being assigned initial replication factors chosen uniformly between 10 and 30. We conducted 5 simulations for each of the settings. The error bars give the standard deviation of the experimental series.

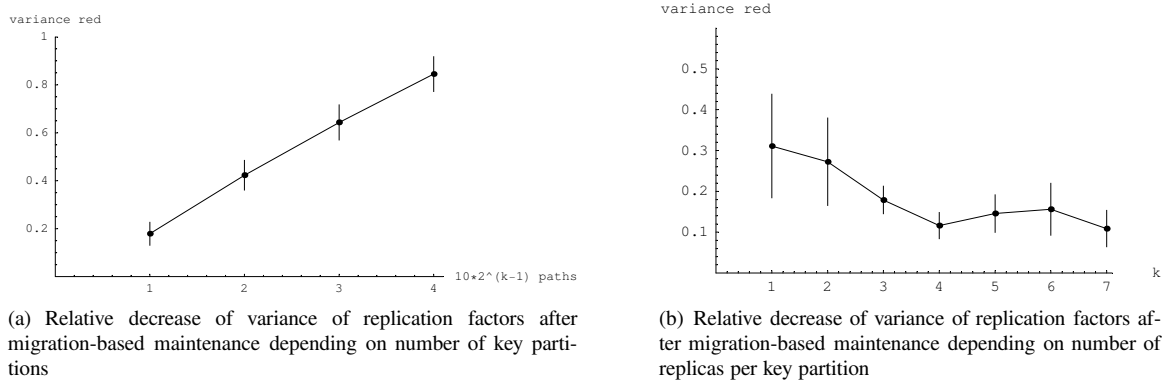


Figure 2: Maintenance of replication load-balance

Figure 2(b) shows the rate of the reduction of variance as a function of different numbers of peers associated with each key partition. We used a P-Grid with  $p = 20$  partitions and assigned to each partition uniformly randomly between  $k$  and  $3k$  peers, such that the average replication factor was  $2k$ . The other settings were as in the previous experiment. Actually variance reduction appears to slightly improve for higher replication factors. This results from the possibility of a more fine-grained adaptation with higher replication factors.

**Simultaneous balancing of storage and replication load in a dynamic setting.** In this experiment we studied the behavior of the system under dynamic changes of the data distribution. Both storage load balancing by restructuring the key partitioning (i.e. extending and retracting paths in P-Grid) and replication balancing by migration were performed simultaneously. We wanted to answer the following two questions:

1. Is the maintenance mechanism adaptive to changing data distributions?
2. Does the combination of restructuring and migration scale for large peer populations?

The experimental setup was: We generated synthetic, unbalanced P-Grids with  $p = 10, 20, 40, 80$  paths and chose replication factors for each path uniformly between 10 and 30. Thus, for example, for  $p = 80$  the expected peer population was 1600. The value  $\delta_{max}$  was set to 50 and the dataset consisted of approximately 3000 unique Zipf-distributed data keys, distributed over the different peers such that each peer held exactly those keys that pertained to its current path. Since the initial key partition is almost uniform the data load of the peers varies considerably, and some peers temporarily hold many more data items than their accepted

maximal storage  $\delta_{max}$  load would be. Then the algorithms for P-Grid restructuring by path extension and retraction and path migrations were executed simultaneously.

Table 3 shows the result of our experiment. We executed an average of 382 rounds in which each peer initiated interleaved restructuring and maintenance operations, which was sufficient for the system to reach an almost steady state.

Number of peers	Number of paths		$R_{\sigma^2}^a$		$D_{\sigma^2}^b$	
	initial	final	initial	final	initial	final
219	10	43	55.47	3.92	180,338	175
461	20	47	46.30	10.77	64,104	156
831	40	50	40.69	45.42	109,656	488
1568	80	62	35.80	48.14	3,837	364

<sup>a</sup> $R_{\sigma^2}$  is the variance of the replication factors for the different paths

<sup>b</sup> $D_{\sigma^2}$  is the variance of the number of data items stored per peer

Table 3: Results of simultaneous balancing

The experiments show that the restructuring of the network as well as replication balancing was effective and scalable:

1. In all cases the data variance dropped significantly, i.e., the key space partitioning properly reflects the (changed) data distribution. Because of the randomized choices of the initial P-Grid structure and the data set, the initial data variance is high and varies highly. It actually depends on the degree to which the randomly chosen P-Grid and the data distribution already matched. From the case  $p = 40$ , we conclude that this has also a substantial impact on the convergence speed since more restructuring has to take place. Actually, after doubling the number of interactions, the replication variance dropped to 20.93, which is an expected value.
2. With increasing number of replicas per key partition the replication variance increases. This is natural as fewer partitions mean higher replication on average and thus higher variance.
3. With increasing peer population the final data variance increases. This is expected as we used a constant number of interactions per peer and the effort of restructuring grows logarithmically with the number of key partitions.

While our algorithms do not need much computation per peer, and hence have low overhead per peer in a real network, simulating the whole system for larger populations takes substantial effort. A single experiment with  $3 * 10^5$  interactions, as was needed to produce the results given in this section, takes currently up to 1 full day of simulation time. Thus we had to limit the number and size of the experiments. Nevertheless they indicate the feasibility, effectiveness and scalability of the algorithms, which have also been implemented in the current version of P-Grid [7].

## 7 Related Work

A detailed overview and classification of current approaches for P2P systems (structured, unstructured, hierarchical) is given in [6]. In the following we focus on the specific issues of load balancing and replication in P2P systems.

For data replication in P2P systems we can distinguish five different methods (partially according to the classification from [16]): *Owner replication* replicates a data object to the peer that has successfully located it through a query (Napster, Gnutella, Kazaa). *Path replication* replicates a data object along the search path that is traversed as part of a search (Freenet, but would also be applicable to unstructured P2P networks in order to replicate data more aggressively). *Random replication* replicates data objects as part of a randomized process. [16] shows that for unstructured networks this is superior to owner and path replication. *Controlled replication* replicates a data object a pre-defined number of times upon insertion (Chord [25], CAN [22], and Pastry [23]). This approach does not adapt replication to the changing environment with variable resource availability. *Adaptive replication* tries to uniformly exploit the storage resources available at peers while also trying to achieve uniform distribution of the replicas of a data object, i.e., for each data object approximately the same number of replicas exist (P-Grid).

Replication of index information is applied in structured and hierarchical P2P networks. For the super-peer approach it has been shown that having multiple replicated super-peers maintaining the same index information increases system performance [26]. Structured P2P networks maintain multiple routing entries to support alternative routing paths if a referenced node fails.

With respect to load balancing in DHT based systems only a few recent works have been reported. The application of uniform hashing and its limited applicability have already been discussed in the introduction.

The load balancing strategy for Chord proposed in [10] uses multiple hash functions instead of only one in the original to select a number of candidate peers. Among those the one with least load stores the data item and the others store pointers to it. However, this scheme does not scale in the number of data items due to the effort incurred by having to maintain the redirection pointers. Moreover, using a predetermined number of hash functions do not give any adaptivity according to the systems requirement. Also Chord's original search no longer works and essentially multiple Chord overlays have to be maintained which are interconnected among themselves in a possibly unpredictable manner.

Another scheme for load balancing for Chord is suggested in [21] based on virtual servers. Nodes are responsible to split the data space to keep the load of each virtual server bounded. The splitting strategy is similar to the splitting used in our storage load balancing strategy, however, this work does not consider the effects on replication nor on search efficiency.

Online load-balancing has been a widely researched area in the distributed systems domain. It has often been modelled as balls into bins [20]. Traditionally randomized mechanisms for load assignment, including load-stealing and load-shedding and power of two choices [18] have been used, some of which can partly be reused in the context of P2P systems as well [10, 11]. In fact, from storage load-balancing perspective, [11] compares closest to our approach because it provides storage load-balancing as well as key order preservation to support range queries, but in doing so, they no more provide any guarantee for searches of isolated keys. As mentioned wherever applicable, load-balancing in DHTs pose several new challenges, which call for new solutions. We need to deal with the dynamic membership (off-online behavior of peers) and dynamic content, and there is neither global coordination nor global information to rely on, and the load-balancing mechanism should ideally not compromise the structural properties and the search efficiency of the DHT, while preserving the semantic information of the data (e.g. lexicographic ordering to enable range searches). While no other adaptive replication mechanism in DHTs have been proposed, by disentangling the peer identifier from the associated keys, we avoid the need to change peer identifiers for storage load balancing, unlike other approaches, like in [17]. The dynamic nature of P2P systems is also different from the online load-balancing of temporary task [9] because of the lack of global knowledge and coordination. Moreover, for replication balancing, there are no real bins, and actually the number of bins vary over time because of storage load balancing, but the balls (peers) themselves have to autonomously migrate to replicate overloaded key spaces. Also for storage load balancing, the balls are essentially already present determined by the data distribution, and it is essentially the bins that have to fit the balls rather than the other way round, by dynamically partitioning the key space.

Substantial work on distributed data access structures has also been performed in the area of distributed databases on scalable data access structures, such as [13, 14]. This work is apparently relevant, but the existing approaches apply to a different physical and application environment. Databases are distributed over a moderate number of fairly stable database servers and workstation clusters. Thus reliability is assumed to be high and replication is used only very selectively [15] for dealing with exceptional errors. Central servers for realizing certain coordination functions in the network are considered as acceptable and execution guarantees are mostly deterministic rather than probabilistic. Distributed search trees [12] are constructed by a full partitioning, not using the principle of scalable replication of routing information at the higher tree levels, as originally published in [19] (with exceptions [27]). Nevertheless, we believe that at the current stage the potential of applying principles developed in this area to P2P systems is not yet fully exploited.

## 8 Conclusions

Existing uncoordinated online load-balancing mechanisms do not address the requirements of DHT-based P2P networks which are in the focus of current P2P research to become the substrate for next generation Internet-scale distributed information systems. We compared the new load-balancing problems of DHT-based P2P systems with the standard model of “balls into bins” so that wherever possible we can apply existing solutions. But more importantly we identified the new and specific requirements of this family of P2P systems, and proposed new algorithms to efficiently and effectively achieve simultaneous storage and replication load-balancing relying only on local information available at each peer. Some of the important novelties of our solution in comparison to other load-balancing mechanisms proposed in the context of P2P systems are: Our mechanism allows the access structure to adapt and restructure dynamically, but preserves the structural properties of the DHT, unlike other mechanisms which require extrinsic mechanisms like redirection pointers, that make queries inefficient. The effort incurred by our load-balancing approach is low because it requires no extra communication but we gather statistic data from normal interactions and “piggy-back” the load-balancing into the standard information exchanges required by the DHT. We also preserve key ordering, which is vital for range queries, and maintain search efficiency irrespective of key distribution. Additionally, unlike some other proposals [17], our solution does not require the peers to change identity, which is important as soon as information on the characteristics of specific peers is exploited and is prerequisite for creating semantic overlay networks as a basic constituent in distributed information management. The approach presented in this paper is implemented in our P-Grid system [7] which is available at <http://www.p-grid.org/>.

## References

- [1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS)*, 2001.
- [2] K. Aberer. Efficient Search in Unbalanced, Randomized Peer-To-Peer Search Trees. Technical Report IC/2002/79, Swiss Federal Institute of Technology, Lausanne (EPFL), 2002. <http://www.p-grid.org/Papers/TR-IC-2002-79.pdf>.
- [3] K. Aberer, A. Datta, and M. Hauswirth. Efficient, self-contained handling of identity in peer-to-peer systems. *IEEE Transactions on Knowledge and Data Engineering*, 2004. To appear.
- [4] K. Aberer, M. Hauswirth, M. Puceva, and R. Schmidt. Improving Data Access in P2P Systems. *IEEE Internet Computing*, 6(1), 2002.
- [5] K. Aberer and M. Puceva. Efficient Search in Structured Peer-to-Peer Systems: Binary v.s. k-ary Unbalanced Tree Structures. In *International Workshop On Databases, Information Systems and Peer-to-Peer Computing. Collocated with VLDB 2003.*, 2003.
- [6] Karl Aberer and Manfred Hauswirth. *Practical Handbook of Internet Computing*, chapter Peer-to-Peer Systems. CRC Press, 2003. To be published.
- [7] Karl Aberer, Manfred Hauswirth, and Roman Schmidt. Experimental evaluation results of an advanced DHT-based peer-to-peer system. Technical Report IC/2004/22, EPFL, 2004. Demonstrator submitted to VLDB04.

- [8] L. Onana Alima, S. El-Ansary, P. Brand, and S. Haridi. DKS(N,k,f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, 2003.
- [9] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. *Journal of Algorithms*, 22:93–110, 1997.
- [10] J. Byers, J. Considine, and M. Mitzenmacher. Simple Load Balancing for Distributed Hash Tables. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.
- [11] D. R. Karger and M. Ruhl. New Algorithms for Load Balancing in Peer-to-Peer Systems, 2003. IRIS Student Workshop (ISW).
- [12] B. Kröll and P. Widmayer. Distributing a Search Tree Among a Growing Number of Processors. In *ACM SIGMOD Conference*, pages 265–276, 1994.
- [13] W. Litwin, M. Neimat, and D. A. Schneider. RP\*: A Family of Order Preserving Scalable Distributed Data Structures. In *VLDB*, pages 342–353, 1994.
- [14] W. Litwin, M. Neimat, and D. A. Schneider. LH\* – A Scalable, Distributed Data Structure. *ACM Transactions on Database Systems*, 21(4):480–525, 1996.
- [15] W. Litwin and T. Schwarz. LH\*RS: A High-Availability Scalable Distributed Data Structure using Reed Solomon Codes. In *SIGMOD Conference*, pages 237–248, 2000.
- [16] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *International Conference on Supercomputing*, 2002.
- [17] G. S. Manku. Randomized ID Selection for Peer-to-Peer Networks. Technical report, Stanford University, 2004. <http://dbpubs.stanford.edu:8090/aux/index-en.html>.
- [18] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.
- [19] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1997.
- [20] M. Raab and A. Steger. “Balls into Bins” - A Simple and Tight Analysis. In *RANDOM*, 1998.
- [21] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load Balancing in Structured P2P Systems. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, LNCS. Springer, 2003.
- [22] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM*, 2001.
- [23] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.
- [24] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup–hypercubes, ontologies, and efficient search on peer-to-peer networks. *LNCS*, 2530, 2003.
- [25] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM*, 2001.
- [26] Beverly Yang and Hector Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.
- [27] H. Yokota, Y. Kanemasa, and J. Miyazaki. Fat-Btree: An Update-Conscious Parallel Directory Structure. In *International Conference on Data Engineering*, pages 448–457, 1999.