EPFL Technical Report IC/2002/79

Efficient Search in Unbalanced, Randomized Peer-To-Peer Search Trees

Karl Aberer
School of Computer and Communication Sciences
Swiss Federal Institute of Technology (EPFL)
1015 Lausanne, Switzerland
Tel: +41-21-6934679, FAX: +41-21-6938115
karl.aberer@epfl.ch

Abstract

Scalable mechanisms to support efficient key-based search in distributed systems are an important part of the infrastructure of peer-to-peer systems and global information systems. They received substantial attention both in information and communication systems research. A particularly important class of approaches is based on a principle of scalable distribution of binary search trees that has been introduced by Plaxton [9]. When adapting the shape of such a tree search structure to the data distribution in order to obtain load balancing, the search trees may become highly unbalanced. We show that for P-Grid, a Plaxton-like distributed search structure that we first introduced in [1], the expected communication cost for searches is strictly limited by $\log(n)$ where n is the number of peers. This result is completely independent of the shape of the underlying tree. The approach exploits the randomization principle of the P-Grid structure by virtue of its decentralized and randomized construction process.

1 Introduction

Scalable indexing mechanisms in distributed environments have been receiving substantial attention in the recent years. Two classes have been studied in the literature: For indexing scalable distributed databases on workstation clusters various variants of distributed search trees and hash-based access structures have been investigated (see e.g. [7]). Usually they are called *scalable distributed data structures (SDDS)* and are characterized by a client-server architecture, a medium number of nodes, and typical by some form of global coordination, such as split coordinators or global directories. For implementing global-scale resource access in a P2P architectures similar structures, so-called *distributed hash tables (DHT)*, have been been studied [1, 2, 3, 4, 9, 10, 11]. DHTs implement routing schemes for quickly locating resources identified by a data key in a network of n peers (typically in $O(\log(n))$ time). The search can be started at any peer, without relying on a centralized directory. They differ from SDDSs through complete (or almost complete) degree of decentralization. The use of routing tables to refine searches in

a stepwise manner works as follows: each peer is associated with an element chosen from the space of data keys and becomes responsible for answering search requests for this data key (and probably a subspace associated with the data key). Peers maintain routing tables that allow to route search requests such that each routing step increases the number of matching bits between the searched data key and the peer key. Therefore a search ends after a maximum number of steps that corresponds to the length of the searched key.

¿From a database perspective we can interpret DHTs as distributed search trees, where each peer maintains information about the path from the root of the tree to the leaf of the tree that corresponds to its peer key. DHTs thus differ from SDDSs by using an increasing degree of replication of the search structure at higher granularity. This principle is accredited to [9], but similar schemes have also been investigated for the implementation of distributed search trees (such as distributed B+-Trees) in parallel and distributed databases [5, 12]. In the following discussion we focus on DHTs.

An important issue for distributed indexing is *load balancing*. A standard approach to associate peers with data keys is to apply a hash function to the peer's IP address (typically extended by some random number). Such a mechanism generates a certain peer key distribution. There is no reason why this distribution should be related in any way to the data key distribution. For example, the peer key distribution could be uniform and the data key distribution could be highly skewed and vice versa.

Therefore highly unevenly distributed workloads for peers in terms of storing data and routing search requests are likely to occur (a problem that is discussed in [4], for example). In the context of database systems this issue has been addressed by developing balanced search structures, such as B+-Trees. Thus, a possible approach is to develop distributed versions of balanced search trees. However, this imposes challenging problems in terms of maintaining the integrity of the trees in the presence of updates, considering the unreliable nature of a typical P2P environment.

In this paper, we pursue a different direction: We build on the P-Grid search structure that has been introduced in [1]. Logically P-Grid is based on a binary trie. This enforces a unique way of constructing the search tree globally, such that peers need not to coordinate at a global level on how the tree is organized (e.g., what split predicate is used at the root node). This allows us to keep processing localized, a crucial requirement in a decentralized peer-to-peer system. Analogous to standard DHT approaches the logical tree is distributed such that each peer holds a complete path from the root of the tree to a leaf of the tree. In contrast to standard DHT approaches we use an adaptive mechanism to associate peers with keys, such that the distribution of peer keys follows the distribution of data keys. Therefore each peer will have the same storage load. In a nutshell, this mechanism is based on mutual interactions among peers in which they perform binary splits of the data key space taking into account the existing data load in the current partitions. The details of this distributed algorithm are presented in [2] and related constructions for the multi-dimensional case have been given in [8].

The details of the construction process are not the focus of this paper, which is on dealing with the complexity of searches on the search structures resulting from the construction process described. As a consequence of the construction process the logical search tree that underlies the search structure organization will be typically unbalanced and processing of search requests

is no longer guaranteed to be efficient (in the worst case linear in n if the tree is linear). It might even occur that we traded storage load balancing for search efficiency.

The important observation is, however, that we consider communication cost, i.e., the number of messages exchanged over the network, as the essential search cost.[1] For example, in the best case the initial peer contacted for answering a request might be the one holding the data object and the search cost is one message. We will show in this paper that this effect generalizes in a way, such that, no matter of how unbalanced the logical tree is (in the worst case even linear in depth), the expected search cost for a search in terms of number of messages required on a P-Grid is logarithmic, more precisely bounded by $\log(n)$. The only requirement is that the P-Grid is uniformly randomly chosen among all possible P-Grids that can result from the randomized construction process for a given data distribution.

The result is reminiscent of the well-known fact that random trees have expected logarithmic depth [6]. However it is structurally different, since the data structures we are considering are distributed and substantially more complex, such that a direct relationship between the results cannot be established. The result we present is of interest for a number of reasons: It illustrates that the changed physical characteristics in distributed and particular communication-intensive P2P architectures (e.g., the predominant role of communciation cost) lead to both interesting new questions and methodological approaches in distributed data management. With P-Grid we propose an abstract model that captures essential characteristics of a number of related approaches to that end. More specifically, we provide a theoretical analysis of an adaptive search data structure for decentralized search. Previous analyses [4] were made for non-adaptive search structures, and search efficiency resulted from the structural properties of the approach (e.g., maximal key length). In contrast, in our approach we exploit randomization for achieving efficiency in data access. In fact, we believe that exploiting randomization is natural, given the unreliable nature of the environment we consider. As we illustrate by our result, randomization is a powerful tool to take advantage of in P2P data management, which is another interesting aspect of our result.

The remainder of the paper is organized as follows. In Section 2 we introduce the P-Grid data access structure and the notations necessary for the subsequent analysis. In Section 3 we introduce the first main result, namely that the expected search cost is $\log(n)$. In Section 4 we provide a worst case analysis, showing that a large deviation of the search cost from the expected average value is rare and we discuss this result under various aspects. We conclude the paper in Section 5 by giving indications on possible further developments of the theory.

2 The P-Grid Access Structure

We assume that a set $\mathcal{S} \subseteq \{0,1\}^*$ is given. The elements of $\mathcal{S}$ are considered as a set of identifiers for peers. Peers store data items that are identified by keys in $\mathcal{K} \subseteq \{0,1\}^*$. We assume that the keys in $\mathcal{K}$ have a length that is at least the maximal length of the elements in $\mathcal{S}$, i.e.,

[1] A standard assumption both for SDDSs and DHTs.

$$\min_{k \in \mathcal{K}} |k| \geq \max_{s \in \mathcal{S}} |s| = s_{max}$$

In order to uniquely associate keys with peers we assume that different elements in $\mathcal{S}$ are not in a prefix relationship.

$$s, s' \in \mathcal{S} \Rightarrow s \not\subseteq s' \wedge s' \not\subseteq s$$

where $s \subseteq s'$ denotes the prefix relationship. We also say that $\mathcal{S}$ is *prefix-free*. A set $\mathcal{S}$ with this property defines uniquely a trie (more precisely a radix-exchange trie), where each leaf node of the trie is associated exactly with one element of $\mathcal{S}$. The definition of the trie generated by $(\mathcal{S})$ can be given as follows:

$$T([t : \mathcal{S}_t]) \quad = \quad \begin{cases} < T([t0 : \mathcal{S}_0)], T([t1 : \mathcal{S}_1]) > & |\mathcal{S}_t| > 1 \\ < t > & \mathcal{S}_t = \{t\} \\ < \epsilon > & |\mathcal{S}_t| = 0 \end{cases}$$

where $t0, t1$ are the strings obtained by appending $0, 1$ to t, $\mathcal{S}_t = \{s \in \mathcal{S} | t \subseteq s\}$ for $t, s \in \{0,1\}^*$. The trie $T([\epsilon : \mathcal{S}_\epsilon)]$, where ϵ is the empty string, is the logical data structure that underlies the construction of P-Grid, our distributed data access structure.

In order to allow every search to successfully terminate we impose a further condition on $\mathcal{S}$. For every prefix $s_1 s_2 \ldots s_l$ of a string $s \in \mathcal{S}$ either there exist $s', s'' \in \mathcal{S}$ such that $s' \in \mathcal{S}_{s_1 s_2 \ldots s_l 0}$ and $s'' \in \mathcal{S}_{s_1 s_2 \ldots s_l 1}$ or $s = s_1 s_2 \ldots s_l$. We also say that $\mathcal{S}$ is *complete*. This leads to the following definition of a P-Grid.

Definition 1. A P-Grid $P \in \mathcal{P}_\mathcal{S}$ for a prefix-free, complete set $\mathcal{S} \subseteq \{0,1\}^*$ is defined by the partial function

$$ref_\mathcal{S}^P : \mathcal{S} \times N \rightarrow \mathcal{S}$$

with the properties

1. $ref_\mathcal{S}^P(s, l)$ is defined for all $s \in \mathcal{S}$ and $l \in N$ with $1 \leq l \leq |s|$
2. $ref_\mathcal{S}^P(s, l) \in \mathcal{S}_{s_1 s_2 \ldots s_{l-1}(1-s_l)}$ where $s = s_1 s_2 \ldots s_{l-1} s_l \ldots s_k, k \geq 0$

Different P-Grids can be constructed depending on the choice of $ref_\mathcal{S}^P(s, l)$. An important observation of which we will make use later relates to the fact that in this definition the choices of $ref_\mathcal{S}^P(s, l)$ are independent of each other for different $s \in \mathcal{S}$ and $l \in N$. Distributed algorithms for constructing such a P-Grid (and thus choosing values for $ref_\mathcal{S}^P(s, l)$) are introduced in [2].

Each peer identified by $s \in \mathcal{S}$ is associated with a location $loc(s)$ (in the network). Searches can start at every peer. Peer s knows the locations of the peers referenced by $ref_\mathcal{S}^P(s, l)$, but not of other peers. The function $ref_\mathcal{S}^P(s, l)$ provides thus the necessary routing information to forward search requests to other peers in case the searched key does not match the peer identifier. The search algorithm is as follows. Let $t \in \mathcal{S}$ be the unique element in $\mathcal{S}$ being a prefix of the

searched data key. t is unique because $\mathcal{S}$ is prefix-free. Let $s \in \mathcal{S}$ be the identifier of the peer where the search starts. We define then the following recursive algorithm.

$$\begin{aligned} search(t, loc(s)) \quad := \quad & \textbf{if } t = s \textbf{ then } return(loc(s)) \textbf{ else} \\ & \{\text{determine } l \text{ such that } t_1 \ldots t_{l-1}(1-t_l) \subseteq s; \\ & search(t, loc(ref_{\mathcal{S}}(s,l)))\} \end{aligned}$$

Theorem 1. The algorithm $search(t, loc(s))$ always terminates successfully.

Proof sketch: Due to the definition of $ref_{\mathcal{S}}$ the function search will always find the location of a peer at which the search can continue (use of completeness). With each invocation of $search(t, loc(s))$ the length of the common prefix of s and t increases at least by one. Therefore the algorithm always terminates.

In a distributed environment the relevant cost measure for an algorithm is the number of messages that are exchanged. Each invocation of $search$ corresponds to forwarding the search task to a different peer. Therefore we define now the search cost in a P-Grid.

Definition 2. The *search cost* of a search in a P-Grid $P \in \mathcal{P}_{\mathcal{S}}$ for a data key $t \in \mathcal{S}$ starting at $s \in \mathcal{S}$ is the number of invocations of the function $search(t, loc(s))$. We denote this cost by $\sigma_P^s(t)$.

We introduce some notations required for the following presentation. For a given $t \in \mathcal{S}$ we denote $\mathcal{S}_j^t = \mathcal{S}_{t_1 t_2 \ldots t_{j-1} t_j}$, $\mathcal{S}_{j^-}^t = \mathcal{S}_{t_1 t_2 \ldots t_{j-1}(1-t_j)}$ and $|t| = d$. For cardinalities we introduce $|\mathcal{S}| = n$, $|\mathcal{S}_j^t| = n_j^t, j \geq 0$, and $|\mathcal{S}_{j^-}^t| = n_{j-1^-}^t, j > 0$. Note that $n_0^t = n$, $n_d^t = 1$, $n_{j^-}^t = n_j^t - n_{j+1}^t, j > 0$, and $\sum_{j=1}^{l} n_{j^-}^t = n - n_l^t$. Figure 1 shows the view on a P-Grid, that a single peer with identifier $t \in \mathcal{S}$ has.

3 Average Search Cost Analysis

The definition of P-Grid does not exclude the case where the depth of the trie $T([\epsilon : \mathcal{S}_\epsilon)]$ is up to linear in the size of $\mathcal{S}$. Therefore searches can require a linear number of messages in the worst case which would make the access structure non-scalable. In the following we show that the expected average search cost is however logarithmic.

Theorem 2. The expected search cost $\sigma_P^s(t)$ for the search of a specific key $t \in \mathcal{S}$ using a P-Grid $P \in \mathcal{P}_{\mathcal{S}}$, that is randomly selected among all possible P-Grids, starting at a randomly selected peer $s \in \mathcal{S}$ is less than $\log(|\mathcal{S}|)$.

Proof: Since the values of $ref_{\mathcal{S}}^P(s,l) \in \mathcal{S}_{s_1 s_2 \ldots s_{l-1}(1-s_l)}$ are independent the set of all possible P-Grids can be given as the product

$$\mathcal{P}_{\mathcal{S}} = \bigotimes_{s \in \mathcal{S}, 0 < l \leq |s|} \mathcal{S}_{s_1 s_2 \ldots s_{l-1}(1-s_l)}$$

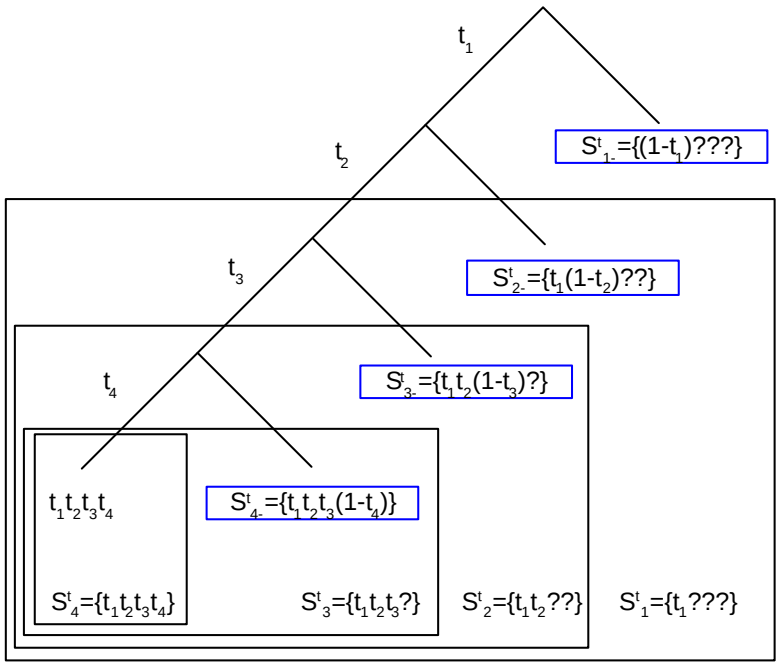

Figure 1: Local view of P-Grid of a peer associated with key $t = t_1t_2t_3t_4$

We assume that references $ref^P_{\mathcal{S}}(s,l)$ are selected with uniform probability from $\mathcal{S}_{s_1s_2\ldots s_{l-1}(1-s_l)}$. Let X^s_l be a uniformly distributed random variable over $\mathcal{S}_{s_1s_2\ldots s_{l-1}(1-s_l)}$. Then the random variable Y gives the probability distribution of P-Grids in $P_{\mathcal{S}}$

$$Y = \bigotimes_{s\in\mathcal{S},1\leq l\leq|s|} X^s_l = \bigotimes_{s\in\mathcal{S},1\leq l\leq|s|} ref^Y_{\mathcal{S}}(s,l)$$

Next we determine the probability distribution of the search cost on a P-Grid. This cost depends on the length of the common prefix of the peer's key at which the search starts and the searched key t. We denote the probability distribution for a given search key t depending on the value of the common prefix length $l, l > 0$ as

$$\sigma_Y^{X^t_l}(t)$$

If we define X^s_0 as a uniformly distributed random variable over $\mathcal{S}$, then $\sigma_Y^{X^t_0}(t)$ denotes the probability distribution of the cost of searches starting at a randomly selected peer. Now we determine the expectation value for $\sigma_Y^{X^t_0}(t)$. In a first step the search will start at a randomly selected peer in $\mathcal{S}$. Thus we have

$$E[\sigma_Y^{X^t_0}(t)] \quad = \quad \sum_{s\in\mathcal{S}} \frac{1}{|\mathcal{S}|} E[\sigma_Y^{ref^Y_{\mathcal{S}}(s,j)}(t)]$$

We distinguish the different classes of peers that can be reached depending on the number of matching bits of the common prefix of the search key and the peer identifier. For $s \in \mathcal{S}_{t_1 t_2 \ldots t_{j-1}(1-t_j)}$ the expected search cost of the remaining search is $E[\sigma_Y^{X_j^t}(t)]$ since it starts at a randomly selected element in $\mathcal{S}_{t_1 t_2 \ldots t_{j-1} t_j} = \mathcal{S}_{s_1 s_2 \ldots s_{j-1}(1-s_j)}$ determined by $ref_{\mathcal{S}}^{P}(Y, j)$. Thus we obtain

$$E[\sigma_Y^{X_0^t}(t)] = \sum_{j=1}^{|t|} Pr[s \in \mathcal{S}_{t_1 t_2 \ldots t_{j-1}(1-t_j)}] E[\sigma_Y^{X_j^t}(t)] = \sum_{j=1}^{|t|} \frac{|\mathcal{S}_{t_1 t_2 \ldots t_{j-1}(1-t_j)}|}{|\mathcal{S}|} E[\sigma_Y^{X_j^t}(t)]$$

We proceed analogously for determining $E[\sigma_Y^{X_l^t}(t)]$. A search starting at a randomly selected element from $\mathcal{S}_{t_1 t_2 \ldots t_{l-1}(1-t_l)}$ for $0 < l < |t|$ is computed as

$$\begin{aligned} E[\sigma_Y^{X_l^t}(t)] &= 1 + \sum_{j=1}^{|t|} Pr[s \in \mathcal{S}_{t_1 t_2 \ldots t_{j-1}(1-t_j)}] E[\sigma_Y^{X_j^t}(t)] \\ &= 1 + \sum_{j=1}^{|t|} \frac{|\mathcal{S}_{t_1 t_2 \ldots t_{j-1}(1-t_j)}|}{|\mathcal{S}_{t_1 t_2 \ldots t_{l-1} t_l}|} E[\sigma_Y^{X_j^t}(t)] \end{aligned}$$

We add 1 to the expected search cost to account for the message used to reach the referenced peer. In the following we denote $E[\sigma_Y^{X_j^t}(t)] = E_j^t$. Since

$$\mathcal{S}_{s_1 s_2 \ldots s_{l-1}(1-s_l)} = \mathcal{S}_{s_1 s_2 \ldots s_{l-1}} \setminus \mathcal{S}_{s_1 s_2 \ldots s_{l-1} s_l}$$

we have

$$|\mathcal{S}_{s_1 s_2 \ldots s_{l-1}(1-s_l)}| = n_{l-1}^t - n_l^t$$

Thus we obtain

$$E_l^t = 1 + \sum_{j=1}^{|t|} \frac{n_{j-1}^t - n_j^t}{n_l^t} E_j^t, 0 \leq l < |t|$$

Computing

$$n_l^t E_l^t - n_{l+1}^t E_{l+1}^t = (n_l^t - n_{l+1}^t) + (n_l^t - n_{l+1}^t) E_{l+1}^t$$

we obtain

$$E_l^t = \frac{n_l^t - n_{l+1}^t}{n_l^t} + E_{l+1}^t$$

We have $E_{|t|}^t = 1$. Therefore we have for the expected search cost

$$\begin{aligned} E_0^t &= \sum_{j=0}^{|t|-1} \frac{n_j^t - n_{j+1}^t}{n_j^t} = \sum_{j=0}^{|t|-1} \int_{n_{j+1}^t}^{n_j^t} \frac{1}{n_j^t}\, dx \\ &\leq \sum_{j=0}^{|t|-1} \int_{n_{j+1}^t}^{n_j^t} \frac{1}{x}\, dx = \int_{n_{|t|}^t}^{n_0^t} \frac{1}{x}\, dx = \int_1^{|\mathcal{S}|} \frac{1}{x}\, dx = \log(|\mathcal{S}|) \end{aligned}$$

q.e.d.

4 Worst Case Analysis

In the following we show that also in the case where the search tree is not of logarithmic depth, the number of P-Grids for which a logarithmic search cost is not achieved is extremely small. To that end we determine the probability that a search for a given P-Grid has reached level l after k steps.

Definition 3. The *search cost* for level l of a search for $t \in \mathcal{S}$ in a P-Grid $P \in \mathcal{P}_\mathcal{S}$ starting at $s \in \mathcal{S}$ is the number of invocations of the function $search(t, loc(s))$ required in order to find a peer in $\mathcal{S}_l^t$. We denote this cost by $\sigma_P^{s,l}(t)$.

Given this definition we denote $p_{l,k}^s(t) = Pr[\sigma_P^{s,l}(t) = k]$. Then our goal is to determine the value of

$$p_{l,k}(t) := \frac{1}{n} \sum_{s \in \mathcal{S}} p_{l,k}^s(t)$$

and in particular $1 - p_{d,k}(t)$, i.e., the probability that the search has not been successful after k steps. First we determine an upper bound for $p_{l,k}(t)$.

Lemma 1. For $k \geq 1$,

$$p_{l,k}(t) \leq \frac{n_{l^-}^t}{n(k-1)!} (\log(n) - \log(n_l^t))^{k-1}$$

assuming $0^0 = 1$. [2] *Proof: see Appendix 1.*

Using Lemma 1 we obtain the following bound on $1 - p_{d,k}(t)$.

Theorem 3. The probability that a search in a P-Grid $P \in \mathcal{P}_\mathcal{S}$ for a key $t \in \mathcal{S}$ of length d starting at a randomly selected peer $s \in \mathcal{S}$ does not succeed after k steps is smaller than $\frac{\log(n)^{k-1}}{(k-1)!}$.

[2] which is a standard assumption in information theory.

Proof:

$$\begin{aligned} 1 - p_{d,k}(t) &= \sum_{i=0}^{d-1} p_{i,k}(t) \leq \sum_{i=0}^{d-1} \frac{n_{i^-}^t}{n(k-1)!} (\log(n) - \log(n_i^t))^{k-1} \\ &\leq \sum_{i=0}^{d-1} \frac{n_{i^-}^t}{n(k-1)!} \log(n)^{k-1} = \frac{\log(n)^{k-1}}{n(k-1)!} \sum_{i=0}^{d-1} n_{i^-}^t \leq \frac{\log(n)^{k-1}}{(k-1)!} \end{aligned}$$

q.e.d.

The given bound is non-trivial in the sense that it is not an immediate consequence of the result on the average search cost. We illustrate this point by an example. A potential distribution of the search cost for the search of strings of length d could be

$$1 - p_{d,k}^H(t) = \frac{\log(n)}{k^2 H(d-1)}$$

where $H(x)$ is the Harmonic Number. One can easily verify that then the average search cost is $\log(n)$ since

$$\sum_{k=1}^{d-1} \frac{k(\log(n))}{k^2 H(d-1)} = log(n)$$

Now $\frac{\log(n)}{k^2 H(d-1)}$ is substantially larger than $\frac{\log(n)^{k-1}}{(k-1)!}$ for large n and for k of order $\log(n)$. For example, if we choose $n = 10^6$, $k = 50$ and $d = 100$, then we obtain $1 - p_{d,k}^H(t) \geq 10^{-3}$ whereas $1 - p_{d,k}(t) \leq 2 \times 10^{-7}$.

One might argue that the result is not relevant in practice as it applies to cases where the maximal depth of the tree is non-logarithmic, and therefore the storage cost for references at certain peers is also non-logarithmic. There are two reasons why this is not necessarily critical: first, storage is abundant, and, more importantly, second, the depth of the tree might be of order $O(\log(n)^d)$, where $d > 1$, or of order $O(n^{\frac{1}{d}})$, where $d > 0$. Both cases are realistic and the bound can give an important improvement as compared to bounding the search cost by the tree depth only.

Finally we can show that the probability of unsuccessful searches drops (more than) exponentially in the number of search steps, as soon as a threshold of $c > \frac{1}{\text{productlog}(\frac{1}{e})} = 3,5911\ldots$ for $k = c\log(n) + 1$ is surpassed.[3]

5 Conclusions

We provided a first theoretical analysis of a distributed search structure suitable for P2P architectures, showing that global control for balancing the search structure in order to bound search

[3] Details are found in Appendix 3

cost is not required when exploiting randomization. There exist a number of natural further developments of this theory by taking into account additional environmental constraints, such as network topologies and communication cost or skewed query distributions. Also the generalization of the approach to structurally different DHTs, for example to n-ary tries, seems to be a promising direction of future research.

Acknowledgements. The author would like to thank Dan Suciu for his insightful suggestions leading to the specification of the P-Grid structure and Manfred Hauswirth for carefully reviewing the manuscript.

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)*, Trento, Italy, 2001.

[2] K. Aberer. Scalable Data Access in P2P Systems Using Unbalanced Search Trees. In *Proceedings of Workshop on Distributed Data and Structures (WDAS-2002), Paris, France*, 2002.

[3] K. Aberer, M. Hauswirth, M. Punceva, and R. Schmidt. Improving Data Access in P2P Systems. *IEEE Internet Computing*, 6(1), Jan./Feb. 2002.

[4] F. Dabek, E. Brunskill, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan. Building peer-to-peer systems with CHORD, a distributed lookup service. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[5] T. Johnson, P. Krishna. *Lazy Updates for Distributed Search Structure* SIGMOD Conference 1993: 337-346

[6] D. Knuth. *Sorting and Searching* The art of Computer Programming, Vol. 2 Addison Wesley, 1973.

[7] W. Litwin, M. Neimat, D. A. Schneider. *RP*: A Family of Order Preserving Scalable Distributed Data Structures.* VLDB 1994: 342-353

[8] A. M. Ouksel, O. Mayer. *The Nested Interpolation Based Grid File.* MFDBS 1991: 173-187

[9] C. G. Plaxton, R. Rajaraman, A. W. Richa. *Accessing Nearby Copies of Replicated Objects in a Distributed Environment*, Proceedings of the 9th Annual Symposium on Parallel Algorithms and Architectures, pp. 311-20, 1997.

[10] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5), 2001.

[11] A. Rowstron, P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for LargeScale Peer-to-Peer Systems 18. Conference on Distributed Systems Platforms, Heidelberg (D), 2001.

[12] H. Yokota, Y. Kanemasa, J. Miyazaki. *Fat-Btree: An Update-Conscious Parallel Directory Structure* ICDE 1999: 448-457

6 Appendix 1: Proof of Lemma 1

Proof: We prove the lemma by induction over k. For $k = 1$ the search starts at a randomly selected peer. The probability that this peer shares with the search string a prefix of exactly length $l-1$ is $\frac{n^t_{l-}}{n}$, and thus

$$p_{l,1}(t) = \frac{n^t_{l-}}{n}$$

Using $0! = 1$ this shows the lemma for $k = 1$.

Now we prove the induction step. If a search has reached level i then it arrived at a peer in S^t_{i-} (see Figure 1). Therefore, we have for $k > 1$

$$p_{l,k}(t) = \sum_{i=0}^{l-1} p_{i,k-1}(t) Pr_{s \in S^t_{i-}}[\sigma^{s,l}_P(t) = k | \sigma^{s,l}_P(t) \geq k-1]$$

Note, that we ignore the fact that $p_{l,k}(t) = 0$ for $l < k$.

If a search has reached level i then it arrived at a peer in S^t_{i-} (see Figure 1). All peers $s \in S^t_{i-}$ have the same probability to reach level l in the next step. They have a uniformly, randomly chosen value for $ref_{\mathcal{S}}(s,i)$ from the set S^t_i. Therefore

$$Pr_{s \in S^t_{i-}}[\sigma^{s,l}_P(t) = k | \sigma^{s,l}_P(t) \geq k-1] = \frac{n^t_{l-}}{n^t_i}$$

Together with the induction hypothesis we thus obtain

$$\begin{aligned}
p_{l,k}(t) &\leq \sum_{i=1}^{l-1} \frac{n^t_{i-} n^t_{l-}}{n^t_i n (k-2)!} (\log(n) - \log(n^t_i))^{(k-2)} \\
&= \frac{n^t_{l-}}{n(k-2)!} \sum_{i=0}^{l-1} \frac{n^t_i - n^t_{i+1}}{n^t_i} (\log(n) - \log(n^t_i))^{(k-2)} \\
&= \frac{n^t_{l-}}{n(k-2)!} \sum_{i=0}^{l-1} \int_{n^t_{i+1}}^{n^t_i} \frac{1}{n^t_i} (\log(n) - \log(n^t_i))^{(k-2)} dx \\
&\leq \frac{n^t_{l-}}{n(k-2)!} \sum_{i=0}^{l-1} \int_{n^t_{i+1}}^{n^t_i} \frac{1}{x} (\log(n) - \log(x))^{(k-2)} dx \\
&= \frac{n^t_{l-}}{n(k-2)!} \int_{n^t_l}^{n} \frac{1}{x} (\log(n) - \log(x))^{(k-2)} dx \\
&= \frac{n^t_{l-}}{n(k-2)!} \frac{1}{k-1} (\log(n) - \log(n^t_l))^{(k-1)}
\end{aligned}$$

This proves the induction hypothesis. *q.e.d.*

7 Appendix 2: A Bound on Unsuccessful Searches

Using Stirling's formula we obtain

$$1 - p_{d,k}(t) \leq \frac{\log(n)^{k-1}}{(k-1)!} \leq \frac{n}{\sqrt{2\pi(k-1)}}(\frac{e\log(n)}{k-1})^{(k-1)}$$

If we set $k = c\log(n) + 1$, then we can derive a bound on the number of search requests that have not been answered within k steps as follows

$$1 - p_{d,k}(t) \leq \frac{1}{\sqrt{2\pi\log(n)}} n^{1+c\log(\frac{e}{c})}$$

Note that the exponent $1 + c\log(\frac{e}{c})$ becomes negative if $c > \frac{1}{\text{productlog}(\frac{1}{e})} = 3,5911\ldots$. For illustration we derive an upper bound that shows that the number of unsuccessful searches drops exponentially fast once this bound is reached. By setting $k = \frac{1}{p}\log(n) + s + 1$ with $p = \text{productlog}(\frac{1}{e})$ we obtain for $n \leq 10^6$ and $s \geq 1$, after some algebraic manipulation, the upper bound

$$1 - p_{d,k}(t) \leq \frac{1}{n^{0.0716626+0.0216004s}\sqrt{2\pi(\frac{1}{p}\log(n) + s))}}$$