Indexing data-oriented overlay networks*

Karl Aberer, Anwitaman Datta, Manfred Hauswirth, Roman Schmidt

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

Abstract

The application of structured overlay networks to implement index structures for data-oriented applications such as peer-to-peer databases or peer-to-peer information retrieval, requires highly efficient approaches for overlay construction, as changing application requirements frequently lead to re-indexing of the data and hence (re-)construction of overlay networks. This problem has so far not been addressed in the literature and thus we describe an approach for the efficient construction of data-oriented, structured overlay networks from scratch in a self-organized way. Standard maintenance algorithms for overlay networks cannot accomplish this efficiently, as they are inherently sequential. Our proposed algorithm is completely decentralized, parallel, and can construct a new overlay network with short latency. At the same time it ensures good load-balancing for skewed data key distributions which result from preserving key order relationships as necessitated by data-oriented applications. We provide both a theoretical analysis of the basic algorithms and a complete system implementation that has been tested on PlanetLab. We use this implementation to support peer-to-peer information retrieval and database applications.

1 Introduction

In standard database systems it is common practice to regularly (re-)index attributes to meet changing requirements and optimize search performance. Recently, structured peer-to-peer overlay networks are increasingly being used as an access structure for highly distributed data-oriented applications, such as relational query processing, metadata search or information retrieval [5, 19]. Their use was motivated by the presence of certain features that are supported by their design such as scalability, decentralized maintenance, and robustness under network churn. Compared to unstructured overlay networks which are also being proposed for these applications [13, 16], structured overlay networks additionally exhibit much lower bandwidth consumption for search.

The standard maintenance model for peer-to-peer overlay networks assumes a dynamic group of peers forming a network where peers can join and leave, essentially in a sequential manner. In addition proactive or reactive maintenance schemes are used to repair inconsistencies resulting from node and network failures or to re-balance load in order to react to data updates. These approaches to maintenance, that have been extensively studied in the literature, correspond essentially to updating database index structures in reaction to updates.

In contrast to this, almost no results exist on how to efficiently construct a large overlay network from scratch, i.e., how to bootstrap a new, large-scale, structured overlay network in a practical way within reasonable time. This is understandable insofar as most of the work on overlay networks was done under the assumption of providing an efficient resource location scheme using an application-specific, yet fairly stable, resource identifier space (e.g., file names for file sharing).

With the increasing adoption of structured overlay network technology for data-oriented applications this assumption no longer holds. Resources are identified by dynamically changing predicates and different overlay networks can be used simultaneously, each of them supporting a specific addressing need. We can illustrate these requirements by a typical application case of peer-to-peer information retrieval which we investigated recently.

The standard application of structured overlay networks in peer-to-peer information retrieval is the implementation of a distributed inverted file structure for efficient keyword based search. In this scenario, several situations occur, in which the overlay network has to be constructed from scratch:

- A set of documents that is distributed among (a potentially very large number) of peers is identified as holding information pertaining to a common topic. To support efficient retrieval for this specific document collection, a dedicated overlay network implementing

*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 and was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project Evergrow No 001935.

**Proceedings of the 31st VLDB Conference,
Trondheim, Norway, 2005**

inverted file access may have to be set up.

- A new indexing method, for example, a new text extraction function for identifying semantically relevant keywords or phrases, is being used to search a set of semantically related documents distributed among a large set of peers. Since the index keys change as a result of changing the indexing method a new overlay network needs to be constructed to support efficient access.
- Due to updates to a distributed document collection an existing distributed inverted file has become obsolete. This may either result from not maintaining the inverted file during document updates or due to changing characteristics of the global vocabulary and thus changing the indexing strategy (e.g., term selection based on inverse document frequency). Thus a complete reconstruction of the overlay network is required.
- Due to catastrophic network failures the standard maintenance mechanisms no longer can reconstruct a consistent overlay network. Thus the overlay networks needs to be constructed from scratch. Of course, this scenario applies generally in any application, but becomes more probable when multiple overlay networks are deployed in parallel.

In principle a (re-)construction of an overlay network in any of these scenarios can be achieved by the standard maintenance model of sequential node joins and leaves. Most existing proposals for structured overlay networks [17, 24, 25] do not offer a completely parallel construction process involving all peers simultaneously. They assume a model of joins of peers in an essentially sequential process. However, this approach encounters two serious problems:

- The peer community will have to decide on a serialization of the process, e.g., electing a peer to initiate the process. Thus the peer community has to solve a leader election problem, which might turn out to be unsolvable for very large peer populations without making strong assumptions on coordination or limiting peer autonomy.
- Since the process is performed essentially in a serialized manner, it incurs a substantial latency. In particular it does not take any advantage of potential parallelization, which would be a natural approach.

In principle some systems like Pastry [24] would support concurrent construction as they take an optimistic approach in which concurrent node joins are possible as long as there are no conflicts. However, this assumes that there already exists a large overlay, so that conflicts are rather unlikely. In an early stage of bootstrapping and with large number of peers joining concurrently, conflicts will be very likely, however. Thus this type of strategy is not applicable to the problem we are addressing. DKS [6] avoids this problem by equipping joining peers with an approximate routing table which in the course of the operation of the overlay will be corrected (correction on use). While this approach is robust, it incurs considerable efforts as on average the number of lookups per peer required to stabilize the network is of the same order as the number of node joins and leaves.

In this paper we will address the problem how a structured overlay network can be constructed efficiently from scratch, a problem that the research community has only recently identified and started to address [2, 8, 14]. Our approach is a generic mechanism to autonomously partition a keyspace in a completely parallel manner. The approach can potentially be used for constructing any structured overlay with fixed key space partitioning [7].

In data-oriented applications there exists an additional factor that adds to the difficulty of finding a solution to this problem: load balancing. When using overlay networks for semantic processing of keys (range queries being a popular example) the canonical method of uniform hashing of keys to remove skew in the key distribution is no more applicable. This has led to substantial research on including load balancing features into overlay networks [2, 12, 17]. During construction this must be taken into account, thus the construction approach also has to solve load balancing problems. In fact, we will address two types of load balancing problems simultaneously: the balancing of storage load among peers under skewed key distributions and the balancing of the number of replica peers across key space partitions. The first problem is important to balance workload among peers and is solved by adapting the overlay network structure to the key distribution. The second one is important to guarantee approximately uniform availability of keys in unreliable networks where peers have potentially low availability. This is a classical "balls into bins" load balancing problem.

Our approach is based on a keyspace bisection process through a completely decentralized, parallel, and randomized algorithm for assigning peers to key space partitions in proportion to the key distributions of the partitions. By recursively applying keyspace bisections, peers can incrementally construct the overlay network while maintaining load balance. We will introduce our approach in the context of the P-Grid overlay network structure [3], which we have developed over the last years, though the essential elements of the approach are applicable to all overlay networks using fixed key space partitioning schemes, such as CAN [23] or Pastry [24]. We demonstrate the theoretical correctness of the basic keyspace bisection process by analysis and simulation and show the feasibility of building a complete system matching the theoretically predicted behavior with experimental results obtained from a full-fledged implementation deployed on the PlanetLab [11] infrastructure. The resulting system (available at http://www.p-grid.org) is currently used to implement both peer-to-peer retrieval (http://www.alvis.info/) and peer-to-peer data management systems [1].

2 Overview of the Approach

2.1 A trie-structured overlay

We assume that data keys are taken from the key space consisting of the interval $[0,1[$. The design of the P-Grid overlay network is based on two simple principal ideas: (1) *Divide and conquer:* The key space is recursively bisected such that the resulting partitions carry approximately the same workload and peers are associated with those partitions. Using a *bisection approach* greatly simplifies decentralized load balancing by local decision making. (2) *Canonical trie structure:* Bisecting the key space induces a canonical trie structure which is used as the basis for implementing a standard, distributed *prefix routing scheme* for efficient search. The resulting overlay is illustrated in Figure 1.

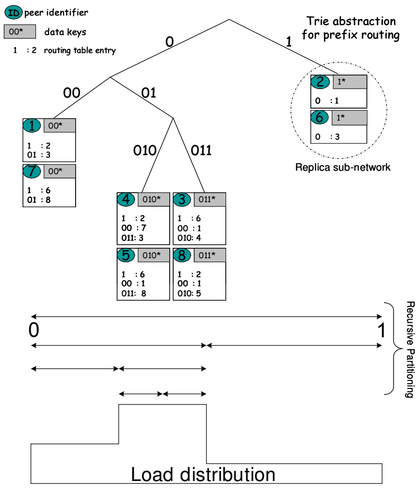


Figure 1: Trie-structured overlay network

At the bottom we see a possible skewed distribution of data keys in the interval $[0, 1[$. We bisect the interval such that each resulting partition carries (approximately) the same load. Each partition can be uniquely identified by a bit sequence. We associate one or more peers—in the example exactly two—with each of the partitions. We will call the bit sequence of a peer's partition the peer's *path*. The bit sequences induce a trie structure which is used to implement prefix routing. Each peer maintains references in its routing table that pertain to its path. More specifically, for each bit position of its path it maintains one or more randomly selected references to a peer that has a path with the opposite bit at this position. Thus the trie structure is represented in a distributed fashion by the routing tables of the peers. This topology is analogous to other prefix routing schemes that have been devised [20, 24] and have been classified as a *fixed key space partitioning* scheme for structured overlay networks in the literature [7]. Search in such an overlay network is performed by resolving a requested key bit by bit. When bits cannot be resolved locally, peers forward the request to a peer from its routing table.

We use replication in two ways in order to increase the resilience of the overlay network when nodes of network links fail. Multiple references are kept in the routing tables, thus providing alternative access paths, and multiple peers are associated with the same key space partitions (*structural replication*) in order to provide data redundancy. Since the routing choices are made by randomly choosing peers from the complementary sub-tree at each level, the resulting overlay network additionally provides efficient search in terms of the communication cost of $O(log(n))$ message, where n is the number of leaf nodes in the tree, irrespective of the shape of the tree [2].

2.2 Overlay Network Construction

The process of constructing such an overlay network from scratch should require low latency, i.e., be highly parallel and require minimal bandwidth consumption. At the same time the following load balancing criteria should be achieved:

1. The partitioning of the search space should be such that each partition holds a maximal data load of d_{max}, e.g., measured as the number of keys present in the partition. We will call d_{max} also the maximal storage load in the following.
2. Each resulting partition should be associated with a constant number of peers n_{min}, such that the availability of the different data keys is approximately the same. We will call n_{min} also the minimal replication factor in the following.

With perfect load balancing these properties can be achieved iff. $d_{tot}n_{min} = d_{max}n$, where d_{tot} is the total number of data keys and n is the number of peers. Algorithm 1 shows our global partitioning algorithm $Partition$ that attempts to achieve these load balancing goals by best effort while bisecting the key space, if the idealizing assumptions are not met.

Algorithm 1 Partition(p, n, d)

```
if $d \geq 2d_{max}$ and $n \geq 2n_{min}$ then
    if $n \frac{min(d_0,d_1)}{d} \geq n_{min}$ then
        $n_0 = n \frac{d_0}{d} n; n_1 = n \frac{d_1}{d}$
        Partition($p_0, n_0, d_0$); Partition($p_1, n_1, d_1$)
    else
        if $d_0 < d_1$ then
            $n_0 = n_{min}; n_1 = n - n_0$
            Partition($p_0, n_0, d_0$); Partition($p_1, n_1, d_1$)
        else
            {analogous}
        end if
    end if
end if
```

The algorithm works as follows. Assume n peers are associated with one key space partition p containing d data keys and two sub-partitions p_0 and p_1 containing d_0 respectively d_1 data keys, such that $d = d_0 + d_1$. To achieve load balance criterion 1, a fraction of $n\ \frac{d_i}{d}$ of peers should be associated with partition p_i for $i = 0, 1$. In case $n\ \frac{d_i}{d} < n_{min}$ at least n_{min} peers should be associated with p_i to achieve load balance criterion 2. $Partition$ recursively applies this bisection step to the key space.

For various reasons this algorithm will achieve the load balancing goals only approximately. Provided the number of data keys is large enough, i.e., $d_{tot} > d_{max}n/n_{min}$, the number of peers associated with a partition will be between n_{min} and $2n_{min} - 1$, instead of constant n_{min}. For very skewed data distributions it can happen that very small partitions contain a large fraction of the data keys, and bisection "disperses" many peers to underloaded partitions even before reaching such partitions. These are fundamental problems of any bisection approach. However, for practical data distributions and large peer populations these problems are more theoretical in nature and $Partition$ achieves good load balancing properties provided n_{min} and d_{max} are chosen properly.

We will use in the following $Partition$ as an algorithm that defines what we consider as an optimal partitioning of the search space among peers and a resulting optimal overlay network. Since in a peer-to-peer system no global coordination exists, the problem we intend to solve is to achieve the partitioning generated by $Partition$ by a decentralized process approximately. We will measure the quality of a solution by determining the deviation from the optimal partitioning.

In a decentralized process peers do not have precise information on the number of peers and keys present in a partition and cannot know which decision the other peers in a partition take with respect to associating themselves with a partition. The only available information is on the set of locally stored data keys and information gathered from local interactions with other peers.

The decentralized process we design is based on random peer encounters and a set of basic local interactions. The random encounters can be initiated by performing random walks on a pre-existing unstructured overlay network. The interactions peers can perform in their encounters can be classified in three categories, as shown in Figure 2.

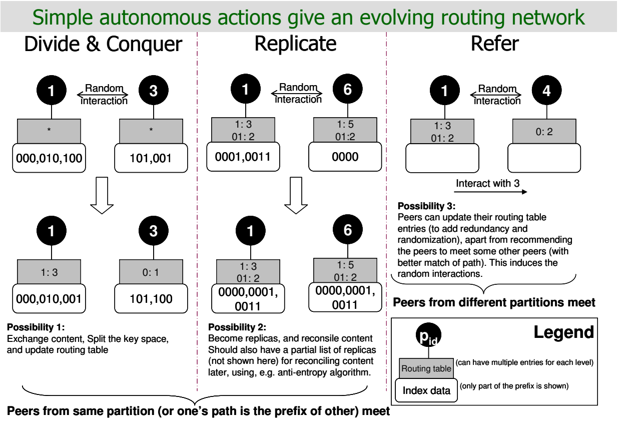

Simple autonomous actions give an evolving routing network
Divide & Conquer
Replicate
Refer
1
Random interaction
3
*
*
000,010,100
101,001
1
3
1: 3
0: 1
000,010,001
101,100
Possibility 1:
Exchange content, Split the key space, and update routing table
1
Random interaction
6
1: 3
01: 2
1: 5
01:2
0001,0011
0000
1
6
1: 3
01: 2
1: 5
01: 2
0000,0001, 0011
0000,0001, 0011
Possibility 2:
Become replicas, and reconsile content
Should also have a partial list of replicas (not shown here) for reconciling content later, using, e.g. anti-entropy algorithm.
Peers from same partition (or one's path is the prefix of other) meet
1
Random interaction
4
1: 3
01: 2
0: 2
Interact with 3
Possibility 3:
Peers can update their routing table entries (to add redundancy and randomization), apart from recommending the peers to meet some other peers (with better match of path). This induces the random interactions.
Peers from different partitions meet
Legend
p_id
Routing table
(can have multiple entries for each level)
Index data
(only part of the prefix is shown)

Figure 2: Network evolution

If peers belong to the same partition they can either $repartition$ the present partition (a divide-and-conquer strategy) or $replicate$ the data keys they currently hold. If they do not belong to the same partition, they can $refer$ each other to other peers using their routing table entries and thus route to a peer that belongs to the same partition.

If peers from the same partition meet, they may decide to $repartition$ in case the current partition contains a sufficient number of data keys to justify a further split, i.e., the partition is overloaded (corresponding to line 1 in $Partition$). They can coordinate locally their decision. In addition, peers keep a reference to the peer encountered after a split, and thus incrementally construct their routing tables.

We can thus reduce the problem of load-balanced overlay network construction to the problem of decentralized partitioning of one key space partition. The problem is that a large number of peers have to perform the decision to split independently for allowing a fast construction of the overlay network, while making these independent decisions in a way that the ratio of the number of peers matches the ratio of the data load in the two partitions. In other words, the global behavior of the distributed decision making process should match the outcome of the partitioning step in the global partitioning algorithm $Partition$ (corresponding to lines 3 and 7 in $Partition$). The solution to this problem is one of the central contributions of the paper and will be discussed in detail in Section 3.

3 Decentralized Partitioning

Consider a set P of $n + 1$ peers which hold data keys from key space K. The space K is partitioned into two parts, 0 and 1, such that the load measured in number of data keys related to the partitions, l_0 and l_1 are p and $1 - p$. In the following we assume w.l.o.g. that $0 \leq p \leq \frac{1}{2}$. Then the partitioning that we would ideally like to achieve should have the following properties:

1. *Proportional replication:* Each peer has to decide for one of the two partitions such that (in expectation) a fraction p of the peers decides for 0 and a fraction $1-p$ for 1. Thus the workload becomes uniformly distributed among the peers, meeting the load-balancing criteria in the resulting overlay.
2. *Referential integrity:* During the process each peer has to encounter at least one peer that decided for the other partition. Thus the peers have the necessary information to construct a routing structure, i.e., the overlay infrastructure, for delegating requests for keys they are no longer associated with.

A peer can initiate interactions with any peer selected uniformly randomly from P. We measure the cost of an algorithm solving the problem in terms of the number of interactions initiated by peers and this cost should be minimized. The quality of an algorithm solving the problem is measured by the deviation of the resulting distribution of peers from an optimal distribution that can be achieved based on global knowledge and coordination. First we assume that the value of p is known to all peers. We will analyze the influence of having only approximate knowledge of p by sampling the locally stored data keys later.

To clarify the critical issues we first discuss two simple heuristic approaches: In the case of $p = \frac{1}{2}$, a simple strategy to adopt would be that peers which have not yet decided for a partition, initiate a random interaction. If the contacted peer is also undecided, the peers decide for different partitions (*balanced split*), otherwise the peer initiating the interaction decides opposite to the contacted peer which has decided already (*unbalanced split*). In this way it learns about a peer from the other partition. Since the algorithm is symmetric, in expectation the same number of peers will decide for each partition, and it provides the best possible performance within the model, since in each interaction every possible decision is taken. We call this strategy *eager partitioning*. While the eager partitioning strategy works well for $p = \frac{1}{2}$, it cannot be employed for other values of p.

For an arbitrary but known p, a possible strategy, which we call *autonomous partitioning* **(AUT)**, would be that each peer makes a decision for one of the two partitions in advance, even without meeting any other peer and then tries to meet some peer from the other partition in order to satisfy the referential integrity constraint. In this setting, obviously some of the peer interactions are "wasted," whenever peers which have decided for the same partition meet. For the specific case of $p = \frac{1}{2}$, by modeling the interactions as Markovian processes, we observed that $2\log(2) = 1.386$ interactions are initiated on an average per peer asymptotically (i.e., for large n), as compared to $\log(2) = 0.693$ interactions per peer with eager partitioning. Thus autonomous partitioning is not an optimal strategy.

3.1 Adaptive eager partitioning

In the following we introduce a method for such an optimized solution to the partitioning problem, that has the characteristics of eager partitioning but works for all p. Due to space constraints we can only summarize the main points of the analysis. However, the full analysis can be found in the long version of this paper [4].

Adaptive eager partitioning **(AEP)** algorithm:

1. Each undecided peer initiates interactions with a uniformly randomly selected peer until a decision is reached. Selecting peers uniformly at random is a non-trivial problem in itself which we solve by a variant of random walks.
2. If the contacted peer is undecided the peers perform a balanced split with probability $0 \leq \alpha(p) \leq 1$ and maintain references to each other.
3. If the contacted peer has already decided for 0 then the contacting peer decides for 1 and maintains a reference to the contacted peer.
4. If the contacted peer has already decided for 1 then the contacting peer decides for 0 with probability $0 \leq \beta(p) \leq 1$ and with probability $1 - \beta(p)$ for 1. In the first case it maintains a reference to the contacted peer. In the second case it obtains a reference to a peer from the other partition from the contacted peer.

It is straightforward to see that condition (2) of the partitioning problem is satisfied. The question is now to determine how to satisfy condition (1) by properly choosing the probabilities $\alpha(p)$ and $\beta(p)$.

We model the peer interactions as a Markovian process using mean value analysis. We assume that in each step i a peer which has not yet found its counterpart contacts another randomly selected peer. By p_i^0 and p_i^1 we denote the number of peers that have decided in step i for 0 and 1, respectively. Initially, $p_0^0 = p_0^1 = 0$. At the end of the process in some step t we have $p_i^0 + p_i^1 = n + 1$. We first assume that $\alpha(p) = 1$. Informally speaking, with this $\alpha(p)$ the partitioning proceeds as fast as possible, optimizing the required number of interactions. Then the model can be given as

$$\begin{aligned} p_i^0 &= p_{i-1}^0 + \frac{1}{n}(n - p_{i-1}^0 - (1-\beta)p_{i-1}^1) \\ p_i^1 &= p_{i-1}^1 + \frac{1}{n}(n - \beta p_{i-1}^1) \end{aligned}$$

To determine the proper value of β for a given value of p, we have to solve this recursive system. The first important observation is that the recursion terminates as soon as no more undecided peers exist, i.e., as soon as $p_i^0 + p_i^1 = n + 1$. Thus we have first to find a value t_β such that $p_{t_\beta}^0 + p_{t_\beta}^1 = n + 1$. In general this will not be an integer value, but in the context of mean value analysis we allow fractional steps. By standard solution methods we obtain

$$\begin{aligned} p_i^0 &= \frac{n}{\beta}(2\beta - 1 + (1 - \frac{\beta}{n})^i - 2\beta(\frac{n-1}{n})^i) \\ p_i^1 &= \frac{n}{\beta}(1 - (1 - \frac{\beta}{n})^i) \end{aligned}$$

and evaluating the termination condition, we obtain

$$t_\beta(n) = \frac{\log(2)}{\log(\frac{n}{n-1})} + 1 \qquad (1)$$

Note, that t_β does not depend on p, and thus the partitioning process requires the same number of interactions among peers independent of the load distribution. By definition $p = \frac{p_{t_\beta}^0}{n+1}$, thus we obtain a relationship between the network size $n + 1$ and the load distribution p with $\beta(p, n)$, the decision probability to be used.

Having $\beta(p, n)$ dependent on n is problematic for two reasons: First the resulting equation is hard to solve, and second, more importantly, n is not necessarily known to the peers. Since we are interested in situations where n is (relatively) large we thus perform an asymptotic analysis. By letting $n \to \infty$ we obtain the following relationship among p and $\beta(p)$

$$p = 1 - \frac{1}{\beta}(1 - 2^{-\beta}) \qquad (2)$$

Positive solutions for $\beta(p)$ cannot be obtained for all values of p. From Equation 2 we derive that positive solutions exist for $p \geq 1 - \log(2)$. This means that the algorithm cannot partition correctly for too highly skewed partitions. Therefore for $0 \leq p < 1 - \log(2)$ we have to pursue a different strategy, by reducing the probability of balanced splits, i.e., $\alpha(p) < 1$.

Through an analogous analysis, by setting $\beta(p) = 0$, we can derive relationships for $\alpha(p)$:

$$t_\alpha(\alpha, n) = \frac{\log(2\alpha)}{\log(n) - \log(1 - 2\,\alpha + n)} + 1 \tag{3}$$

and for the relation between $\alpha(p)$ and p when $n \to \infty$

$$p = -\frac{\alpha\;(1 - 2\,\alpha + \log(2\alpha))}{(1 - 2\,\alpha)^2} \tag{4}$$

Before we continue with the discussion of different partitioning algorithms, a statement on the modeling approach is necessary: We use a sequential approach to model and analyze what is a concurrent process. This is a simplification as well as an appropriate approximation for our purpose. Assume that the latency in one interaction is such that c other interactions among peers occur concurrently. Then the concurrent behavior of N peers corresponds (approximately) to the sequential behavior of $\frac{N}{c} = n + 1$ groups. The analysis we perform shows that the models we use are sufficiently accurate for relatively small n. Thus for large numbers of peers the model is a sufficiently good approximation, whereas for small N concurrency is less likely to occur and less critical.

3.2 Error Analysis

Up to now we assumed that the value of p is known to all peers. Practically peers will derive an estimate for p by sampling. Therefore, in the following we analyze the effect of errors introduced by only approximate knowledge of p. Other potential sources of errors, such as taking the limit case $n \to \infty$ and using mean value analysis turned out to have a negligible influence.

Assume peers obtain s samples from their locally stored data keys. The samples correspond to Bernoulli variables $X_1, \dots, X_s$ with probability p. The peers estimate p by computing the mean value $X = \frac{1}{s}\sum_{j=1}^{s} X_j$ which is binomially distributed. We would like to determine the effect of an error in estimating p on the values of $\alpha(p)$ and $\beta(p)$ and the resulting effect on the partitioning process when using approximate values of α and β. In the following we will use α and β instead of $\alpha(p)$ and $\beta(p)$ as long as the meaning is clear.

We provide an exemplary error analysis for the evolution of p_i^1 for the case where $p \geq 1 - \log(2)$ since this is algebraically the simplest case. Analogous analysis have been done for the other case, but they are substantially more complex.

We assume that in step i the estimation value $p_i^* = p + \gamma_i$ is used to determine an estimation value $\beta_i = \beta + \epsilon_i$. The error γ_i is the sampling error obtained by the peer initiating step i. Let us denote by $\tilde{p}_i^1 = p_i^1 + \delta_i^1$ the error introduced into the result of the partitioning process due to sampling errors. We can derive the following closed-form expression for δ_i^1 from analyzing the Markov model of the process.

$$\delta_i^1 = \left(1 - \frac{\beta}{n}\right)^i \sum_{j=1}^{i} -\frac{\left(1 - \left(1 - \frac{\beta}{n}\right)^{-j}\right) n\,\epsilon_j}{\beta\;(\beta - n)} \tag{5}$$

Since the sampling errors are presumably small we use a Taylor series expansion to approximate $\beta(p)$. In fact, for reasons that will become clear later, we need to make a second order approximation to perform a proper error analysis. For a given value p, we have

$$\epsilon_i \approx \beta'(p)\gamma_i + \frac{1}{2}\beta''(p)\gamma_i^2 \tag{6}$$

for small γ_i. We now determine the expectation value and standard deviation for δ_t^1 (to simplify the presentation we will write t instead of t_β in the following). Since $E[\gamma_i] = 0$ and $E[\gamma_i^2] = var[\gamma_i] = \frac{1}{s}p(1-p)$ we obtain for the expectation value using (5)

$$E[\delta_t^1] \;=\; \frac{1}{2}\beta''(p)\,\frac{n}{s}p(1-p)\,g(\beta, n, t) \tag{7}$$

where $-0.21 \leq g(\beta, n, t) \leq -0.20$. This shows that sampling introduces a systematic shift of the balance between the resulting partitions. In a concrete implementation we will have to compensate for this systematic error, as will be discussed in more detail subsequently.

Since $var[\sum_{i=1}^{t}\gamma_i] = t\,var[\gamma_i] = \frac{t}{s}p(1-p)$ we obtain for the standard deviation by a similar analysis

$$\sigma[\delta_t^1] \;=\; \beta'(p)\,n\sqrt{\frac{t}{s}p(1-p)}\,f(\beta, n, t) \tag{8}$$

where $0.07 \leq f(\beta, n, t) \leq 0.09$.

The impact of the errors depends in particular also on the behavior of the functions $\beta'(p)$ and $\beta''(p)$. Using numerical differentiation we observed that the functions are well-behaved in the relevant region.

Performing an analogous analysis for $p < 1 - \log(2)$ the behavior of the functions $\alpha'(p)$ and $\alpha''(p)$ will be relevant for the error behavior. We have included a plot of $\alpha''(p)$ in order to point out an important observation (Figure 3): For very small values of p the second derivative grows extremely fast, and consequently the error will be large as well.

The error analysis shows that in the presence of sampling errors, we have to include correction terms in the probabilities $\alpha(p)$ and $\beta(p)$ used in AEP.

$$\alpha_{corr}(p) \;=\; \alpha(p) - \frac{1}{2}\alpha''(p)\frac{1}{s}p(1-p) \tag{9}$$

$$\beta_{corr}(p) \;=\; \beta(p) - \frac{1}{2}\beta''(p)\frac{1}{s}p(1-p) \tag{10}$$

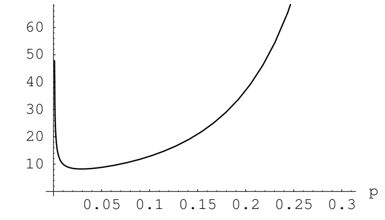


Figure 3: Numerical Solution for $\alpha''(p)$.

3.3 Numerical Simulation of the Markov Model

To validate the correctness of our analytical models we performed numerical simulation experiments. We simulated five models:

1. MVA: simulation of the mean value model for AEP with known p
2. SAM: simulation of the mean value model for AEP; the value of p is estimated from s samples
3. AEP: discrete Simulation of AEP with peers taking discrete decisions based on $\alpha(p)$ and $\beta(p)$ instead of adding mean value contributions as in the mean value model
4. COR: discrete Simulation of AEP with corrected probabilities $\alpha_{corr}(p)$ and $\beta_{corr}(p)$
5. AUT (Discrete Autonomous Partitioning): Discrete simulation of autonomous decision making where p is estimated from s samples

We present the results for $n = 1000$ and $s = 50$. Each experiment has been repeated 100 times.

Figure 4 shows the deviation of the mean value of p_0^t from the expected value $p\,n$ averaged over all experiments. As expected, using sampling for estimating p leads to a systematic deviation of the resulting distribution (SAM, AEP). The error correction strategy (COR) eliminates the deviation almost completely. Clearly, autonomous partitioning (AUT) on average achieves the desired distribution.

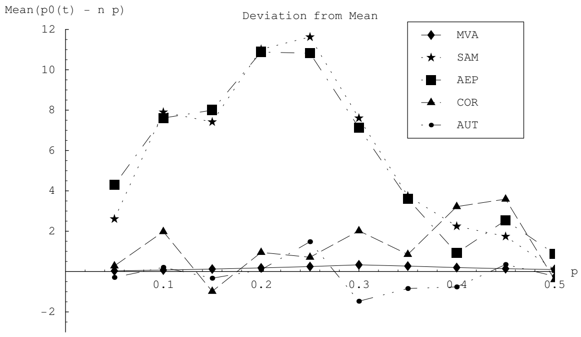


Figure 4: Mean of p_t^0 over 100 experiments, the expected value $p\,n$ is subtracted to highlight the deviation.

Figure 5 shows the cost of each algorithm measured in number of interactions. As theoretically predicted, we observe that adaptive eager partitioning performs better than AUT, except for small values of p (approx. $p < 0.15$) independent of which version is considered (MVA, SAM, COR).

Further experiments with different sample sizes showed that the sample size has practically no influence. Even very small samples (1 or 2 samples) lead to the same results as larger sample sizes. Experiments also showed that adaptive eager partitioning has a further advantage over autonomous partitioning as it reduces the standard deviation of the error in partitioning by approximately a factor of 2. Thus our AEP approach optimizes both performance in terms of number of required interactions and error control in terms of matching the partitioning ratio p.

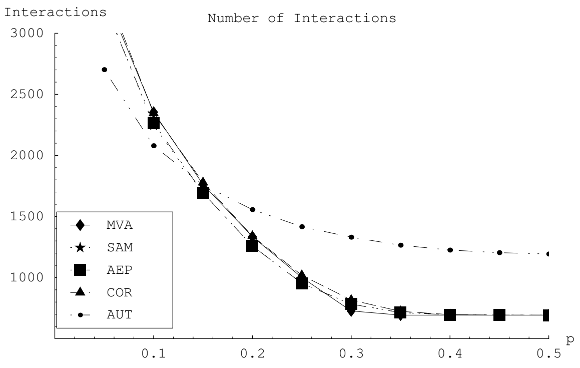


Figure 5: Mean total number of interactions over 100 experiments

Superficially, AEP appears to be a more complex algorithm than AUT while not considerably outperforming AUT. However, the complexity is in the analysis required to determine the correct decision probabilities, whereas for practical implementation AEP has even advantages since it provides an invariant: When taking a decision for a partition, the availability of a reference is guaranteed.

We would like to point out that the problem studied in this section is a novel load distribution problem in the area of distributed systems, particularly because of the referential integrity constraint. A solution to this problem can be useful beyond overlay network construction as we use it here, but also in resource and task distribution and decentralized load-balancing in general.

4 Algorithmic Issues

In order to use AEP for implementing the *Partition* algorithm in a decentralized fashion we have to address several issues related to the global organization of the indexing process.

4.1 Initiating the Indexing Process

In the absence of global coordination the mechanism to reach a decision to initiate the indexing process is not obvious. While it is not the focus of this paper, and the initiation process is orthogonal to the index evolution process, we nonetheless describe a simple, decentralized strategy.

Depending on locally observed queries, individual peers may make autonomous decisions on whether a new index may be necessary or re-indexing may be required. Any of the peers that locally decide that indexing is useful can initiate a vote, by flooding the peer network. This flooding can use the pre-existing, generic, unstructured overlay network which we assume to exist.

When peers receive a voting request they can reply back their local decision. Additionally, helpful information, such as locally available storage space that the peer is willing to contribute to store information for the new index and the number of local data items to be indexed can be piggy-backed. Votes are sent back along the paths they arrived, and multiple votes are aggregated while flowing back to reduce bandwidth consumption. Based on the number of positive and negative responses, the peer which initiated the voting can then decide whether to initiate index construction or not, and can flood the decision back to all peers. Additionally, based on the aggregate storage space available, and the amount of storage required for all the data items (references) in the system, the decision will contain the parameters for ensuring optimized utilization of the available resources and for synchronization of the indexing process. We assume a collaborative environment where the majority of peers does not behave maliciously or in a Byzantine manner, and adheres to the democratic decision of the group, and thus participates in the indexing irrespective of their individual votes.

4.2 Synchronizing and Terminating the Indexing Process

The partitioning algorithm introduced in Section 2 enables reaching a decision in parallel on bisecting the key space proportionally among a group of autonomous peers. In the indexing process the algorithm is executed multiple times and a synchronization mechanism is needed. In addition peers need to autonomously recognize when to terminate the indexing process. We realize this as follows.

The peer communicating the decision to start the indexing process provides the parameters d_{max} and n_{min} as used in $Partition$. The values are chosen such that $d_{max} = d_{avg}n_{min}/2$, where d_{avg} is the average number of data keys peers hold (as mentioned in Section 4.1 this information can be derived from information piggy-backed to the votes). Additionally, it provides a time t_{init}. Before starting to partition, peers replicate their data keys at time t_{init} to n_{min} randomly chosen other peers. Thus at the start of the indexing process all data keys are already replicated the desired number of times in the network.

Besides estimating the number of data keys in the current partition, peers also have to estimate the number of current peers, in order to perform the proper decisions in algorithm $Partition$. Attempting this directly, by learning about all existing replicas at each level of the partitioning process, would unnecessarily slow down the progress of indexing. Instead, we estimate the number of replicas in a partition by analyzing the overlap in the sets of data keys of two peers interacting in a balanced split. If D_i denotes the set of data keys peers $p_i, i = 1, 2$ hold, and $D = D_1 \cup D_2$, then $\frac{|D_1||D_2|n_{min}}{|D|d_{max}}$ is a maximum likelihood estimate of the expected number of peers in the current partition. For example, if $D_1 = D_2$ and $|D_1| = d_{max}$ then it should be expected to have n_{min} peers in the partition since initially data keys have been replicated n_{min} times. To ensure the correctness of this estimation was the purpose of initially replicating the data.

During partitioning, peers that have extended their paths attempt to immediately contact other peers to perform the partitioning at the next level. If they do not succeed in identifying a different peer in the same partition with which a useful interaction can take place, i.e., "divide and conquer" or "replicate", after a fixed number of attempts (e.g., 2), using the *refer* interaction (see Figure 2), they stop to initiate interactions and only will continue after being contacted by another peer. In this way peers that are "ahead of the crowd", e.g., due to faster network connections, are forced to wait for the slower ones. The same mechanism also eventually leads to termination of the process, when peers encounter only fully synchronized copies of themselves.

4.3 Complexity

The goal of our approach to index construction is to perform it with low bandwidth consumption and low latency. With regard to bandwidth consumption a necessary requirement is to perform no worse than a sequential approach using standard construction mechanisms, i.e., $O(n\ log^2 n)$. To study this, we look at the complexity in the case of a balanced key distribution ($p = \frac{1}{2}$). Then for partitioning at one level, peers engage in $log(2)$ bilateral interactions on average. In addition to locating a peer in the same partition at level k, peers have to route on expectation $log(k)/2$ steps when performing the *refer* interaction. This shows that the total number of interactions is also of order $O(n\ log^2 n)$. However, the latency is $O(log^2 n)$ as opposed to $O(n)$ in the standard maintenance model.

4.4 Simulation of the System

To study the global behavior of the indexing algorithms when integrating all the elements discussed so far, we performed simulation studies implemented in Mathematica. We were mainly interested whether the desired load balancing properties would be achieved under the various approximations and whether the algorithm performs as predicted.

In the simulations we used peer populations of sizes 256, 512, and 1024. As data distributions we used a uniform distribution, a Pareto distribution with PDF $\frac{a\ k^a}{x^{1+a}}$ with parameters $k = 1$ and $a = 0.5, 1.0, 1.5$, and a Normal distribution with mean value $\frac{1}{2}$ and standard deviation 0.0513, and test data from text retrieval experiments (project Alvis). In Figure 6 these distributions are denoted as *U*, *P0.5*, *P1*, *P1.5*, *N* and *A*. The Pareto and Normal distributions represent cases with extremely skewed distributions. Initially, we randomly assigned 10 keys from the distributions to peers, so that they held samples. We tested with $n_{min} = 5$ and $n_{min} = 10$ such that at least 5 (respectively 10) replicas of the keys are generated. Typically the experiments had $d_{max} = 10n_{min}$. All experiments were repeated 10 times and the results were averaged. The algorithms were implemented as described above. The experiments were executed on a workstation cluster using up to 36 machines and were running for more than a week. Note that there were 36 separate experiments, each conducted 10 times. Furthermore, in a real network the peers would use exclusive resources, and thus the actual overlay construction process is much faster.

For evaluating the experiments we primarily were determining the degree to which the load balancing of peers across key space partitions worked. To do so, we compared the generated key sets to the distribution, that would be generated by global coordination ($Partition$ algorithm).

The $Partition$ algorithm generates a distribution $(k_i, n_i), i = 1, \ldots, K$, where k_i are the K partitions of the key space generated and n_i are the number of peers associated with each partition. We compared this distribution to the distribution (k_i^d, n_i^d) generated by the decentralized algorithm.

$$\frac{\sqrt{\sum_i^K (n_i - n_i^d)^2}}{\frac{1}{K}\sum_i^K n_i^d}$$

As explained in Section 2, we consider the distribution generated by $Partition$ as the optimal distribution. Measuring the distance to this distribution provides a measure for the quality of load balancing.

The first experiment (Fig 6(a)) for $n_{min} = 5$ and $d_{max} = 10$ shows the quality of load balancing depending on the peer population size for the different distributions. One can observe that the quality remains practically stable independent of the size.

We also investigated the influence of the replication factor n_{min} by comparing $n_{min} = 5, 10, 15, 20, 25$ (Fig 6(b)). In principle the load balancing properties should not be affected as we measure deviations relative to the average replication. This is confirmed for less skewed distributions, whereas for the strongly skewed distributions a certain degradation can be observed. We have still to investigate in detail the reasons for this effect, but most likely it is related to the relatively low number of partitions with high replication factors.

We were also interested in the influence of the sample size d_{max} on the quality of load balancing. It might be expected that more samples lead to higher accuracy. In fact, the result (Fig 6(c)) shows that no such influence exists. This is insofar important as it shows that the partitioning can be done using very small samples which enables several possibilities for optimization to reduce bandwidth consumption.

In order to understand the quality of the load distributions achieved we also analyzed the role of our theoretical framework (Fig 6(d)). We replaced the functions $\alpha_{corr}(p)$ and $\beta_{corr}(p)$ by heuristic functions which likely would be chosen in the absence of a theoretical understanding of their properties. The hypothesis we wanted to verify was whether the concrete nature of these functions plays a significant role in view of the many approximations made in the overall distributed algorithm. We chose

$$\alpha_{heur}(p) = \frac{1}{\frac{1}{p} - 1}, \beta_{heur}(p) = 0$$

These functions exhibit qualitatively the same behavior as the ones used by AEP. The experiment was executed for $n = 256$ and $n_{min} = 5$. The conclusion is clear from the result: Even a minor change to the theoretically correct functions degrades the quality of load balancing substantially. Thus the theoretical basis proves valuable despite many idealizing assumptions.

We also analyzed the communication costs of the algorithm. We can see that both the number of interactions per peer (Fig 6(e)), and the overall bandwidth consumption per peer measured in terms of the total number of data keys exchanged among all peers during the interactions (Fig 6(f)) grow gracefully in terms of the network size, as expected from theory. However, skew in the data distribution can significantly increase the bandwidth consumption.

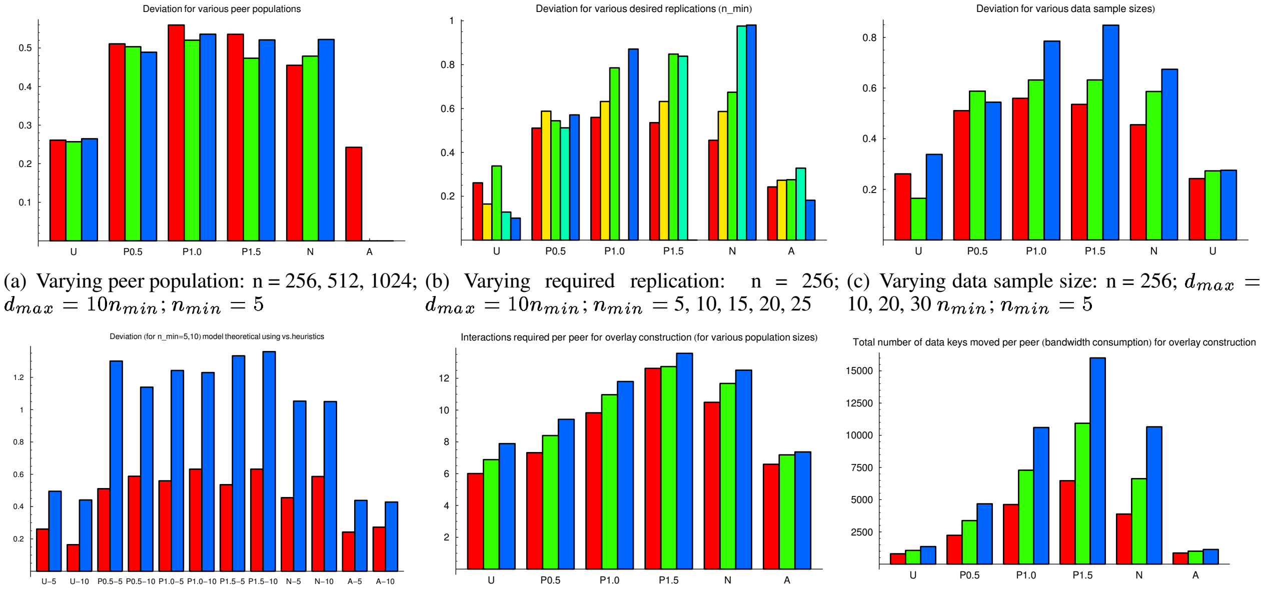
5 Experimental evaluation

We used the PlanetLab infrastructure [11] to obtain results from large-scale experiments under realistic networking conditions and to verify our theoretical predictions and simulation experiments. PlanetLab (http://www.planet-lab.org/) is a global testbed for large-scale experiments with distributed systems. At the moment it consists of approximately 530 nodes geographically distributed over the whole planet running a modified version of Linux to support efficient administration and resource sharing for large-scale experiments. Nodes are connected via a diverse collection of links. Our experiments on PlanetLab ran on up to 300 nodes depending on the number of available nodes. Each node executed one instance of a P-Grid node. When interpreting the results presented in the following, it is important to consider that PlanetLab is shared by a large number of research groups for experiments that are executed in parallel and thus mutually influence the performance considerably especially with respect to absolute latency.

5.1 Experimental setup

We deployed the P-Grid software, i.e., the peers, on all available nodes at the times the experiments were conducted and assigned 10 keys from a real text collection (taken from our Alvis information retrieval project) to each peer. This relatively low number of keys was chosen to speed up experiments and as we have already seen, sample size has little influence on load balancing. To validate our experiments, we also performed tests with larger numbers (up to 2000 keys per peer) and used various distributions, including uniform random distribution and Pareto distribution.

The time-line of the experiments was as follows: In an initial phase starting at time t, peers join the system by contacting a bootstrap peer (until $t + 30min$) and form an unstructured overlay network (from t until $t + 45min$) which is used later to replicate data a fixed number of times (from $t + 45min$ until $t + 60min$). In the replication phase peers randomly choose 5 peers from the unstructured overlay network to replicate their data. Subsequently, from $t + 60min$ to $t + 300min$, the structured overlay network is constructed using the approach presented in this paper. We were especially interested in evaluating the bandwidth consumption during this phase and to verify whether the theoretically predicted load balancing properties of the algorithm are achieved under realistic networking conditions. Then we run queries on the constructed overlay network ($t+300min$ to $t+400min$) to analyze search performance. Each peer performed a search every 1–2 minutes. In the final phase ($t + 400min$ to $t + 500min$) network churn is simulated to evaluate the failure resilience of P-Grid. Each peer independently decides to go offline 1–5 minutes every 5–10 minutes which causes considerable churn that the system has to compensate.



(a) Varying peer population: n = 256, 512, 1024; $d_{max} = 10n_{min}$; $n_{min} = 5$

(b) Varying required replication: n = 256; $d_{max} = 10n_{min}$; $n_{min} = 5, 10, 15, 20, 25$

(c) Varying data sample size: n = 256; $d_{max} = 10, 20, 30\ n_{min}$; $n_{min} = 5$

(d) Theory vs. Heuristics

(e) Interactions per peer: n = 256, 512, 1024; $d_{max} = 10n_{min}$; $n_{min} = 5$

(f) Bandwidth consumed (data keys moved): n = 256, 512, 1024; $d_{max} = 10n_{min}$; $n_{min} = 5$

Figure 6: Simulation results for various experiment scenarios.

5.2 Experimental Evaluation

We first verified that the system behavior matches the theoretical predictions and the simulations. The experiment was performed with 296 peers and compared to simulation results using the same number of peers and the same key set.

The quality of load balancing is evaluated as defined in Section 4.4 and is practically identical for simulations and experiments, with an average of 0.38 for 10 simulations (the standard deviation is 0.05) resp. a value of 0.39 for the experiment. This indicates that the theoretically predicted load distribution properties are met quite accurately by the implementation even under realistic network conditions with slow connections and communication failures.

We now report some system measurements that we made to evaluate the performance of the overlay network, both during the construction phase, as well as in its operational lifetime both in a static situation (no change in peer population) as well as under churn (peers leave and join the network).

Figure 7 shows the number of peers in the overlay at a given time. We see how first peers join the network and the number of peers in the network increases to the maximal number. Then during the construction phase this number is stable (approx. 300 peers) while decreasing again in the final phase where we simulate network churn and a substantial dynamic fraction of peers becomes unavailable.

Figure 8 shows the aggregate bandwidth consumption of all peers (maintenance and queries) in Bytes/sec. During the construction phase the bandwidth consumption reaches a peak of 250 Bytes/sec per peer. The maintenance consumption decreases quickly down to less than 100 Bytes/sec and becomes negligible compared to the bandwidth consumed by queries.

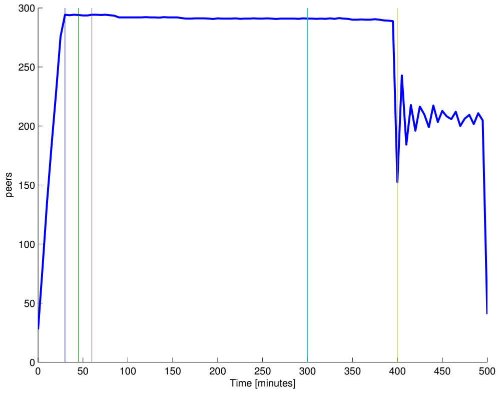


Figure 7: Number of participating peers

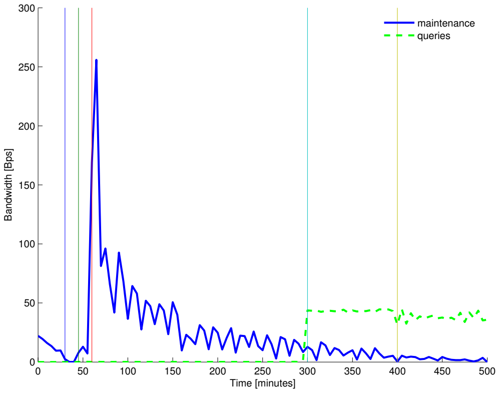


Figure 8: Aggregate bandwidth consumption

Figure 9 shows the average query latency and its standard deviation. The absolute values are relatively high and essentially reflect the poor response time of PlanetLab nodes. The response time is slightly higher with a larger deviation during the network churn because requested peers may be offline which has to be compensated.

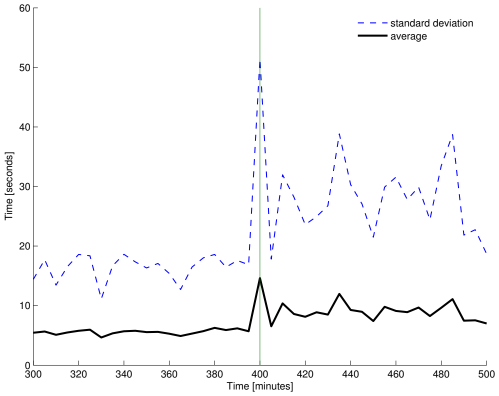


Figure 9: Query latency

We observed that the number of query hops per query is as low as theoretically expected, i.e, approx. half of the mean path length, even during churn. The average path length was slightly below 6 and the average number of query hops per query was approximately 3. Moreover after the construction phase has led to full evolution of the overlay network, all peers discovered all their replicas, and the system had an expected mean replication factor of 5, as intended, and success rate for queries was between 95% and 100% even during network churn. Queries were mainly unsuccessful because of network problems such as lost or corrupted messages.

Finally, we would like to point out that the current experimental evaluation is still limited in the following sense: The moderate number of available peers does not allow us to obtain significant results on the reduction of latency during bootstrapping as predicted by our theoretical analysis in Section 4.3 and which is one of the main properties of our approach.

6 Related work

The fundamental problems to address for any large-scale distributed indexing systems are distributed index construction and load-balancing. Traditionally structured overlay networks, mainly based on distributed hash tables (DHTs), have followed sequential construction and maintenance strategies (online balancing) [12, 17, 24, 25]. In contrast to this, our approach applies a highly parallel strategy which speeds up the construction process, takes advantage of the distributed computing resources by allowing the participants to work independently and asynchronously on the construction, and enables the merging of independently created indices.

To address load-balancing, the standard strategy of overlay approaches is to use uniform hashing of keys to remove skew from the distribution. However, this defeats the applicability of overlay networks to semantic processing of keys (range queries, etc.). Thus in standard overlay approaches, typically an additional index on top of the overlay network needs to be created [22]. The advantage of this approach is its universal usability on top of any DHT. However, it is considerably less efficient than our approach since semantically close data items are not necessarily stored close to each other in the overlay network (high fragmentation), and hence, multiple overlay network queries are required to locate all the semantically close content. Thus, apart from the additional effort of constructing an additional index, such schemes additionally suffer from inefficiencies throughout the operational phase of the system.

In contrast to that, we build a trie that clusters semantically close data, thus realizing *in-network indexing* which enables more efficient query processing. This comes at the expense of a more sophisticated construction process for such data-oriented overlay networks. Additionally, more complex online load-balancing strategies have to be applied, as presented in this paper.

Online load-balancing is widely researched area in the distributed systems domain which often been modeled as "balls into bins" [21]. 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 [10, 15], but with limited applicability. For example, [15] provides storage load-balancing as well as key order preservation to support range queries, but at the cost that efficient searches of isolated keys can no longer be guaranteed.

The dynamic nature of P2P systems is also different from the online load-balancing of temporary tasks [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 varies 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 determined by the data distribution, and it is essentially the bins that have to fit the balls by dynamically partitioning the key space, rather than the other way round.

A distinguishing property of our approach to all other related load-balancing strategies is actually that we address two, sometimes conflicting load-balancing problems—storage load, i.e., balancing the amount of storage used at the nodes, and replication load, i.e., ensuring approximately uniform data availability by having roughly the same number of replicas per data partition. The first step in that direction was a heuristic key space bisection proposal [2]. In comparison to the heuristics, we now exhaustively analyze and refine the bisection mechanism, in order to better understand and guarantee superior load-balancing characteristics in the overlay network emerging from the recursive use of the bisection algorithm. Additionally, we now not only simulate the construction process, but verify the analytically predicted properties using a fully-fledged implementation (P-Grid), deployed on PlanetLab, to back up our analysis and simulation results with large-scale experimental data. The overlay network is already used as a substrate for two data-oriented applications—a peer-to-peer search engine (http://www.alvis.info/) and a semantic overlay network [1].

Furthermore, most existing load-balancing as well as overlay network construction mechanisms have so far been sequential. However, the need for faster overlay construction has recently generated interest in the research community, as is evident from some recent publications [8, 14].

Both [8] and [14] use random interactions among peers, induced potentially by the original unstructured topology, and try to build a desired topology, by essentially trying to sort the peers according to their identifiers that are generated at the beginning of the process. These mechanisms can again be used for overlay networks construction which support search of keys generated by uniform hashing, since then peer identifiers can be simply generated using uniform hashing, as there is no skew in the load-distribution. However, for data-oriented applications such a mechanism has a critical limitation, since peers are predestined for the amount of load (based on the whole set of peer identifiers generated at the beginning of the process), and there is no flexibility or adaptivity for load-balancing, particularly if the load is skewed. Our scheme—this paper as well as [2]—on the other hand adaptively creates the key space partitions and assigns peers to these partitions based on load characteristics, and is thus a more generic parallel overlay construction mechanism. For the special case of uniform load distribution (as it is traditionally assumed in DHTs using uniform hashing), we can easily construct a load-balanced overlay by requiring $p = 0.5$ in each step of the partitioning.

7 Conclusions

The fast (re-)construction of data-oriented structured overlay networks is an emerging research topic which has not yet been covered exhaustively in the literature. (Re-)indexing due to changing application requirements is a frequent scenario in data-oriented applications and necessitates the efficient (re-)construction of overlay networks. Existing approaches are essentially serialized and do not take into account inherent intricacies like preservation of key-ordering relationships to enable semantic processing on data keys. In this paper we have presented an efficient, completely decentralized algorithm which supports the fast, parallel construction of structured overlay networks from scratch based on a recursive bisection scheme that preserves key semantics and provides good load-balancing for skewed distributions both for storage and replication load. We prove the efficiency of our approach by analytical results which are verified by simulation and large-scale experiments of a complete system implementation on PlanetLab. The implementation is available from http://www.p-grid.org/.

References

[1] K. Aberer, P. Cudré-Mauroux, M. Hauswirth, and T. van Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In *ISWC*, 2004.

[2] K. Aberer, A. Datta, and M. Hauswirth. Multifaceted Simultaneous Load Balancing in DHT-based P2P systems: A new game with old balls and bins. *Self-* Properties in Complex Information Systems, "Hot Topics" series, LNCS*, 2005.

[3] Karl Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *CoopIS*, 2001.

[4] Karl Aberer, Anwitaman Datta, Manfred Hauswirth, and Roman Schmidt. Indexing data-oriented overlay networks. Technical Report IC/2005/008, Ecole Polytechnique Fédérale de Lausanne (EPFL), 2005.

[5] S. Abiteboul, I. Manolescu, and N. Preda. Constructing and Querying Peer-to-Peer Warehouses of XML Resources. In *SWDB*, 2004.

[6] 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.

[7] L. Onana Alima, A. Ghodsi, and S. Haridi. A Framework for Structured Peer-to-Peer Overlay Networks. In *Post-proceedings of the Global Computing Conference*, LNCS. Springer Verlag, 2004.

[8] D. Angluin, J. Aspnes, J. Chen, Y. Wu, and Y. Yin. Fast construction of overlay networks. In *SPAA*, 2005.

[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 *IPTPS*, 2003.

[11] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3), July 2003.

[12] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. In *VLDB*, 2004.

[13] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer Data Management System. *TKDE*, 16(7), 2004.

[14] M. Jelasity and O. Babaoglu. T-Man: Gossip-based overlay topology management. In *Engineering Self-Organising Applications (ESOA'05)*, 2005.

[15] D. R. Karger and M. Ruhl. New Algorithms for Load Balancing in Peer-to-Peer Systems, 2003. IRIS Student Workshop (ISW).

[16] G. Koloniari and E. Pitoura. Content-Based Routing of Path Queries in Peer-to-Peer Systems. In *EDBT*, 2004.

[17] G. S. Manku. Balanced binary trees for ID management and load balance in distributed hash tables. In *ACM PODC*, 2004.

[18] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10), 2001.

[19] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing strategies for RDF-based peer-to-peer networks. *J. Web Sem.*, 1(2), 2004.

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

[21] M. Raab and A. Steger. "Balls into Bins" - A Simple and Tight Analysis. In *RANDOM*, 1998.

[22] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief Announcement: Prefix Hash Tree. In *ACM PODC*, 2004.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *ACM SIGCOMM*, 2001.

[24] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[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 ACM SIGCOMM*, 2001.