Cloud, Grid and High Performance Computing:

Emerging Applications

Emmanuel Udoh
Indiana Institute of Technology, USA

Information Science
REFERENCE

Senior Editorial Director: Kristin Klinger
Editorial Director: Lindsay Johnston
Director of Book Publications: Julia Mosemann
Acquisitions Editor: Erika Carter
Development Editor: Hannah Abelbeck
Production Editor: Sean Woznicki
Typesetters: Michael Brehm, Keith Glazewski, Milan Vracarich, Jr.
Print Coordinator: Jamie Snavely
Cover Design: Nick Newcomer

Published in the United States of America by
in (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data
Cloud, grid and high performance computing: emerging applications / Emmanuel Udoh, editor.
p. cm.
Includes bibliographical references and index.
Summary: "This book offers new and established perspectives on architectures, services and the resulting impact of emerging computing technologies, including investigation of practical and theoretical issues in the related fields of grid, cloud, and high performance computing"--Provided by publisher.
ISBN 978-1-60960-603-9 (hardcover) -- ISBN 978-1-60960-604-6 (ebook) 1. Cloud computing. 2. Computational grids (Computer systems) 3. Software architecture. 4. Computer software--Development. I. Udoh, Emmanuel, 1960-
QA76.585.C586 2011
004.67'8--dc22
2011013282

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 21
A Decentralized Directory Service for Peer-to-Peer-Based Telephony

Fabian Stäber
Siemens Corporate Technology, Germany

Gerald Kunzmann
Technische Universität München, Germany[1]

Jörg P. Müller
Clausthal University of Technology, Germany

ABSTRACT

IP telephony has long been one of the most widely used applications of the peer-to-peer paradigm. Hardware phones with built-in peer-to-peer stacks are used to enable IP telephony in closed networks at large company sites, while the wide adoption of smart phones provides the infrastructure for software applications enabling ubiquitous Internet-scale IP-telephony.

Decentralized peer-to-peer systems fit well as the underlying infrastructure for IP-telephony, as they provide the scalability for a large number of participants, and are able to handle the limited storage and bandwidth capabilities on the clients. We studied a commercial peer-to-peer-based decentralized communication platform supporting video communication, voice communication, instant messaging, et cetera. One of the requirements of the communication platform is the implementation of a user directory, allowing users to search for other participants. In this chapter, we present the Extended Prefix Hash Tree algorithm that enables the implementation of a user directory on top of the peer-to-peer communication platform in a fully decentralized way. We evaluate the performance of the algorithm with a real-world phone book. The results can be transferred to other scenarios where support for range queries is needed in combination with the decentralization, self-organization, and resilience of an underlying peer-to-peer infrastructure.

DOI: 10.4018/978-1-60960-603-9.ch021

INTRODUCTION

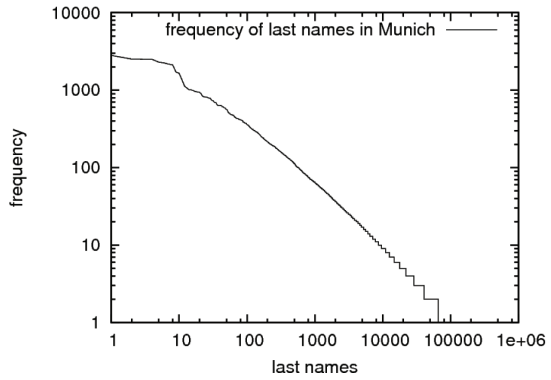
Structured peer-to-peer overlay protocols such as Chord (Stoica et al 2001) are increasingly used as part of robust and scalable decentralized infrastructures for communication platforms. For instance, users connect to an overlay network to publish their current IP address and port number using a unique user identifier as the keyword. In order to establish a communication channel to a user, the user's identifier must be looked up in order to learn the TCP/IP connection data. Registration and lookup of addresses are realized using Distributed Hashtables (DHT).

However, users in such applications do not always know the unique identifier of the person to be contacted. Therefore, it must be possible to look up the identifier in a phone-book-like user directory. When looking up an identifier, the user might not know all data necessary to start an exact query. For example, the user might know the last name of the person to be searched, but not its first name or address. Moreover, people often are not willing to fill out all data fields, e.g. the address of the person to be called. Therefore, the phone book is required to support range queries, like queries for all people with a certain last name.

A challenge arises from the non-uniform distribution of people's names. Figure 1 shows the frequency of last names in the city of Munich, Germany. Last names are Pareto-distributed, or Zipf-distributed, i.e., there are a few last names that are very common, while most last names are very rare.

In this article, we propose the use of Extended Prefix Hash Trees (EPHTs) as a scalable indexing infrastructure to support range queries on top of Distributed Hash Tables. The EPHT is evaluated by using real-world phone book data; experiments show that our approach enables efficient distributed phone book applications in a reliable way, without the need for centralized index servers. A comparison with related work shows that this has not been possible using techniques introduced before.

Figure 1. Frequency of last names in Munich, Germany



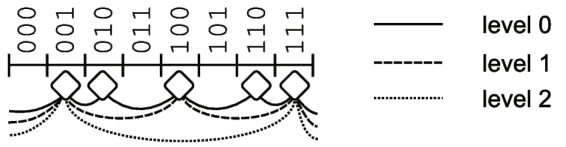
In the following section, we review related work and highlight the problems with current approaches. Then, we present the EPHT algorithm, and compare it with the original Prefix Hash Tree (PHT) algorithm. Then, we evaluate its performance by running a series of experiments. Finally, we summarize our results and show our conclusions.

RELATED WORK

When entries are stored in a Distributed Hash Table, the location of an entry is defined by the hash value of its identifier. A common way to achieve a uniform distribution of the entries among the peers in the DHT is to require the hash function used to calculate the hash value to operate in the Random Oracle Model (Bellare et al, 1993), i.e. even if two identifiers differ only in a single Bit, the hash values of these identifiers are two independent uniformly distributed random variables.

While this hash function allows for good balancing of the data load in a DHT, it makes range queries very costly. Iterating among a range of identifiers that are lexicographically next to each other means addressing nodes in a random order

Figure 2. Skip graph



in the peer-to-peer network. A way to accelerate range queries is to abandon the Random Oracle Model, and to store the entries in lexicographical order. In this section, we discuss three approaches relying on this idea: Skip Graphs (Aspnes et al, 2003), Squid (Schmidt et al, 2004), and Mercury (Bharambe et al, 2004). We point out the difficulties arising with these approaches in scenarios like a distributed phone book.

A comparison between EPHTs and the original PHT algorithm (Rambhadran et al, 2004) is presented after we introduced the EPHT.

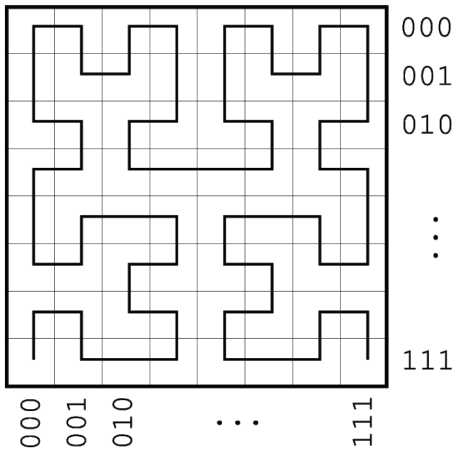
Skip Graphs

Figure 2 shows a linear three-Bit identifier space. The peers, as indicated by diamonds, are randomly distributed among the identifiers. Each peer is responsible for the identifiers in the range between itself and its predecessor or successor.

As shown in Figure 2, Skip Graphs introduce several levels of linked lists for traversing the peers. The higher the level of the list, the more peers are skipped, accelerating routing to specific ranges. By maintaining several independent lists on each level in parallel, Skip Graphs provide balancing of the traffic load and resilience to node failure.

However, the problem with Skip Graphs is that the entries' identifiers are not distributed uniformly among the linked list, while the peers are randomly distributed. Entries for a last name starting with 'S' are very common in the German phone book, while last names starting with 'Y' are very uncommon. Therefore, the peer being responsible for a common entry becomes a hot spot in terms of network traffic and data load.

Figure 3. Squid



Squid

Squid (Schmidt et al, 2004) is an approach for combining several keywords when determining the position of an entry in the Distributed Hash Table. Squid is based on Locality-Preserving Hashing (Indyk et al, 1997), in which adjacent points in a multi-dimensional domain are mapped to nearly-adjacent points in a one-dimensional range.

For example, in a distributed phone book application, one could use a two-dimensional keyword domain, where one dimension is the entries' last name, and the other dimension is the entries' first name. Figure 3 shows how two dimensions can be mapped on a one-dimensional range using a Space Filling Curve (SFC). The SFC passes each combination of the two identifiers exactly once. If the user wants to search for all entries with a last name starting with 'ST' and a first name starting with 'F', then the user simply needs to query the parts of the SFC that lie on the intersection of these two prefixes in the two-dimensional space.

However, as with Skip Graphs, it turns out that the distribution of names in a phone book results in combinations that are very common, while other combinations are very rare. Again, the peers being responsible for common combinations become hot spots in terms of data storage and

traffic load. This could be avoided with Squid by introducing many dimensions in order to distribute the entries among many different peers. But introducing many dimensions results in a tangled-up SFC. As a result, many short fragments of the curve need to be processed for each keyword that is not specified in a query. We evaluated Squid and found that this results in a very high number of peers to be queried in order to find an entry.

Mercury

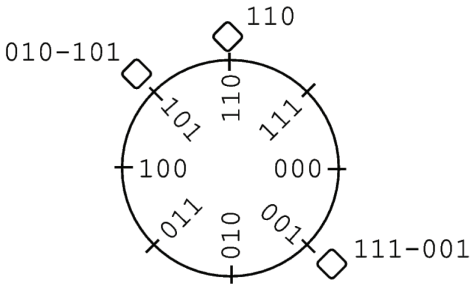
Like Squid, the Mercury approach (Bharambe et al, 2004) supports multi-dimensional keywords. Each dimension is handled within a separate hub, which is a ring-shaped formation of peers. An example of a Mercury hub is illustrated in Figure 4.

The ID range within a hub is ordered linearly, which results in the same load balancing problems as with the other approaches. However, Mercury suggests that peers are moved around dynamically to balance the load. Although this might be a reasonable approach in other scenarios, this raises difficulties in the distributed phone book scenario. First, there are a few very popular last names. A peer being responsible for one of these popular last names cannot be relieved by moving around other peers, and it will stay a hot spot in terms of data load. Second, if peers may choose their position in the overlay deliberately, this raises certain security issues, because an attacker who wants to make a person unreachable can position its peer in a way that it becomes responsible for routing queries to the victim's entry.

Fusion Dictionary

Fusion Dictionaries (Liu et al, 2004) are not a distributed search index, but a load balancing technique that can be combined with search indexes. The idea is to maintain a blacklist of names that are very common, and to cache blacklist entries in large parts of the DHT. If a user queries a last name that is in the blacklist, the query is interrupted and the user is asked to specify the query more precisely, e.g. by including the first name in the query.

Figure 4. Mercury hub



That way, peers being responsible for frequent names are relieved. As the last names are Zipf-distributed, there are only a few names to be included in the blacklist in order to achieve significant load balancing.

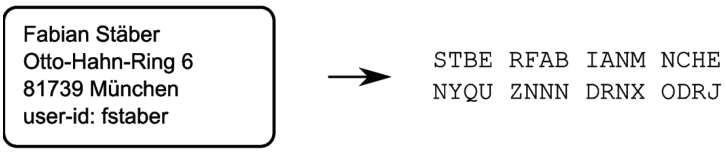
However, in spite of the load balancing achieved with fusion dictionaries, the approaches introduced above still do not fulfill the scalability and performance requirements of large scale communication platforms. In this article, we present the EPHT, which is a search index that does not result in overloaded peers. That way it is unnecessary to introduce additional load balancing techniques.

Summary

The brief survey of related work showed that there are several difficulties with previous range query solutions when applied in the distributed phone book scenario. A more detailed overview of search methods in peer-to-peer systems can be found in (Risson et al, 2006). An analysis of arbitrary search in structured peer-to-peer systems was published in (Hautakorpi et al, 2010).

Approaches supporting real multi-dimensional keywords like Mercury and Squid have the problem of hot spots with very popular last names. Additionally, approaches relying on linear keywords instead of real hashing suffer from overloaded peers being responsible for popular prefixes. In

Figure 5. Generating a 32 char identifier for an entry



Squid, the hot spots in terms of data load could be avoided, but as a trade-off this results in a large number of peers to be queried to find an entry.

In the following section, we introduce the Extended Prefix Hash Tree as a way of enabling efficient range queries, while preserving the advantages of the Random Oracle Model for hashing, which results in a balanced distribution of the entries among the peers in the DHT.

EXTENDED PREFIX HASH TREE ALGORITHM

Each entry in the distributed phone book is associated with an identifier. The identifier is a fixed-length string, consisting of the capital characters [A-Z]. Identifiers are built by concatenating keywords from the entry. In the example in Figure 5, we used the keywords last name, first name, and city.

The order of the keywords determines the relevance of these keywords for range queries. If identifiers are built as in Figure 5, it is possible to search for the last name without knowing the city, but it is not possible to search for the city without knowing the last name. This corresponds to the hierarchical structure of printed phone books, where entries are ordered by city, last name, first name, etc. In order to allow alternative keyword orders, the application must maintain several trees in parallel.

Special characters like whitespaces or the German ä, ö, ü, ß are omitted. That way, both German names "Müller" and "Möller" map into the same string "MLLER". It is up to the application layer to filter out the right results when a user searched for "Müller".

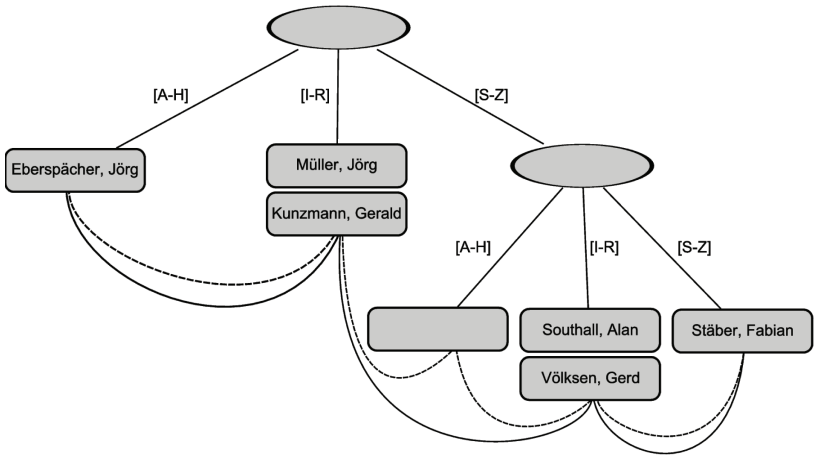
The identifier length must be sufficient to ensure that a unique identifier can be built for each entry with high probability. In our evaluation, the identifiers were 32 characters long. Identifiers that are longer than that are truncated; identifiers that are shorter are padded with random characters.

Growing the Tree

The structure of an EPHT is shown in Figure 6. There are two parameters that determine the shape of the tree:

1. *n* is the number of children per node. Each edge is labeled with a character set, like [S-Z]. The partitioning of the alphabet into character sets is fixed and globally known, and cannot be changed dynamically during runtime. *n* is the number of character sets, which can be any number between 2 and 26. In the section on evaluation, we show that the best performance is achieved with n=26.
2. *m* is the maximum load of the root node, i.e. the maximum number of entries that can be stored on the root node. If the root node exceeds its maximum load, it splits up into *n* child nodes and distributes all entries among the children. The maximum load of each child equals the maximum load of the parent node plus one. The reason for incrementing the maximum load is to prevent recursive splits, if all entries happen to be stored on the same child. If a child node's prefix length (see below for the definition

Figure 6. Example of an extended prefix hash tree with n=3 and m=2



of prefix) equals the identifier length for the entries, then that node cannot split any further, and its maximum load becomes infinite. In the section on evaluation we show that m=100 is a good value.

Each node of the tree is stored as a resource in a DHT, using a hash function operating in the Random Oracle Model. The keyword to be hashed is the prefix of that node in the EPHT, i.e. the sequence of character sets on the path from the root node to the node to be stored. For example, the keyword of the leaf node holding the entry 'Gerd Völksen' in Figure 6 would be '[S-Z][I-R]'. New entries are stored on the leaf node that has the closest matching prefix for the identifier of that entry.

Once a node is split, it becomes an inner node. Inner nodes are kept in the system to indicate the existence of child nodes, but they do not store any data. In particular, inner nodes do not need to store links to their children.

Maintaining the Linked Lists

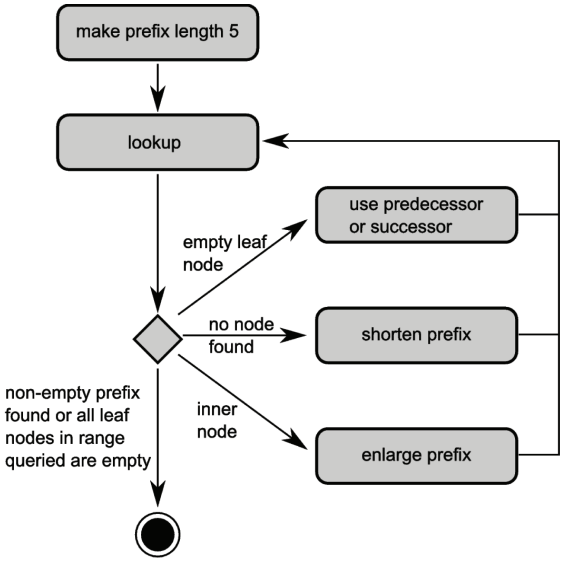
In addition to the tree structure itself, two doubly linked lists are maintained: one for traversing the non-empty leaf nodes, and the other connecting all leaf nodes, including the empty ones. Each element in a list stores the prefix of its predecessor and successor. The linked list is updated upon the following events:

1. A leaf node splits up into child nodes. In that case, the old leaf node must leave the linked lists, the non-empty new child nodes must join the linked list for non-empty nodes, and all new leaf nodes must join the linked list connecting all leaf nodes. The new nodes learn about their initial successors and predecessors from their parent node.
2. An entry is added to a previously empty leaf node. In that case, that node must join the list for non-empty leaf nodes. The node finds its predecessor and successor using the list connecting all leaf nodes.

Performing Range Queries

Usually, tree algorithms imply that nodes are searched starting at the root node and traversing down the tree to a leaf node. This would mean that the peer holding the root node becomes a bottleneck and single point of failure in a distributed tree structure. EPHTs allow lookups to address

Figure 7. Addressing nodes



arbitrary nodes directly, using the prefix of the node as the keyword in the DHT.

Range queries are implemented as follows: First, the issuer of a query finds a random, non-empty leaf node lying somewhere in the queried range. Second, the issuer traverses the linked list of non-empty leaf nodes to the left and to the right, subsequently querying the predecessors and successors, until all matching entries are retrieved.

Figure 7 shows how an initial non-empty leaf node is found that can be used as a starting point for traversing the linked list. We exemplify this using a search for all people with the last name 'Olpp'.

Make Prefix Length 5. The first step is to pad the search string with random characters, and to take the first five characters as an initial prefix to start with. In the example, the initial prefix would be OLPPD. In the section on evaluation we will show why 5 is a good initial prefix length.

Lookup. When this prefix is looked up in the DHT, there are four possible results:

1. A node with that prefix exists and is a non-empty leaf node. In that case, the initial node for traversing the linked list is found.
2. A node with that prefix exists and is an empty leaf node. In that case, the issuer of the query starts traversing the linked list until a non-empty member is found. If all prefixes in the range queried are empty, then the search was unsuccessful.
3. A node with that prefix exists but is an inner node. In that case, the prefix was underspecified, and it must be enlarged by one character. In the example, the next search string might be OLPPDH.
4. There is no node with that prefix. This means the prefix was over-specified, and it must be shortened by one character. In the example, the shortened prefix would be OLPP.

In order to decrease latency, the search can be initialized with several different random paddings in parallel. That way, the linked list can be traversed starting from different positions at the same time.

Removing Entries

Entries do not need to be deleted explicitly. Each entry is associated with a lease time. If it is not renewed within that time, it is deleted. That way, users who are no longer part of the system will be removed after some time.

In EPHTs, once a node has split and become an inner node, this node stays an inner node for ever, even if all entries in its sub-tree have timed-out. That means that the EPHT can only grow, but never shrink. This property is in accordance with our use case, as shrinking the tree would only make sense if the service provider operating the distributed phone book application would permanently loose a significant number of customers, or if the distribution of the name's prefixes changes significantly. Both scenarios happen very slowly, and it is feasible to roll out a software update in that case that will built a new tree from scratch. The persistence of inner nodes enables us to implement extensive caching.

Caching

As the EPHT never shrinks, inner nodes are immutable. They will never be deleted or altered. That means that inner nodes can be cached infinitely in the DHT. Whenever a peer learns about the existence of an inner node, it can cache that information and respond when that prefix is queried the next time. Without caching, prefixes that are accessed very frequently would cause a lot of network traffic for the peer being responsible for that prefix. Using caching, this network traffic can be balanced in the DHT.

COMPARISON WITH THE ORIGINAL PREFIX HASH TREES

The Extended Prefix Hash Tree algorithm presented here derives from the Prefix Hash Tree (PHT) algorithm proposed in (Rambhadran et al, 2004). However, the original PHT could not have been used to implement a distributed phone book without the changes presented in this article. The novelty of our work is twofold:

1. The original PHT is a binary tree enabling Bit-wise processing of keywords. Its design does not support caching, and it handles multiple keywords using a Squid-like approach. This does not match the requirements found in the distributed phone book scenario. Therefore, we extended the PHT in several respects, as described below.
2. The EPHT algorithm has several configuration parameters, like the number of child nodes, and the maximum load of a node. We evaluated the Extended PHT with real-world phone book data, and showed how to gain the best performance.

In the rest of this section, we will show the major differences between the EPHT and the PHT algorithm.

- The original PHT is a binary tree. As shown in the evaluation, binary trees do not scale well in a distributed phone book scenario. Therefore, the EPHT is an n-ary tree, and we recommend to use n=26, i.e. the size of the applied alphabet.
- In the original PHT, if the number of entries in a subtree falls below a certain threshold, that subtree collapses into a single leaf node. The EPHT can only grow, but never shrink, which enables us to introduce extensive caching of inner nodes.
- Empty nodes are not handled specially in the PHT algorithm. In the Extended PHT, we introduced an additional linked list skipping the empty nodes to improve performance. This is because we observed that a significant number of prefixes do never appear in user's names, which results in empty leaf nodes for these prefixes.
- The original PHT proposes to handle multiple keywords using Locality-Preserving Hashing, as in Squid. In our application, we simply concatenate the keywords according to their priority, and pad the result with random data.

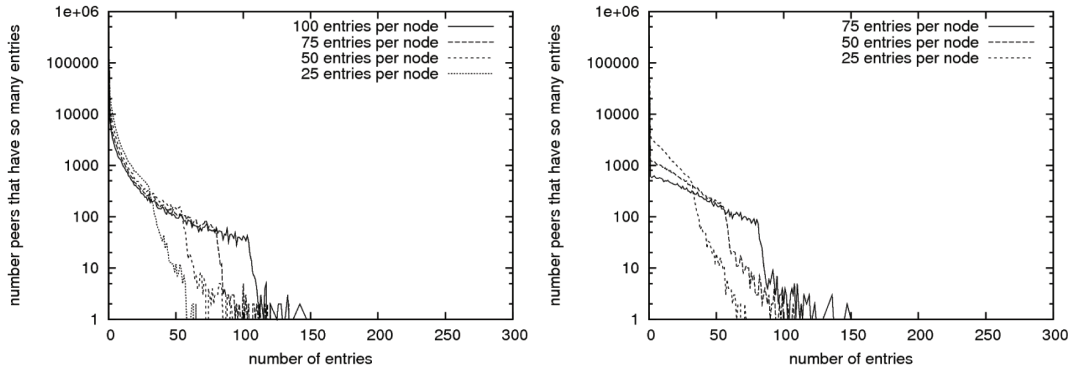
EVALUATION

In this section, we present the simulation results. The evaluation data is taken from a German phone book CDROM from 1997, because newer electronic phone books restrict data export due to privacy regulations. We used the entries for the city of Munich, which has 620,853 entries. As each peer is supposed to provide only its own entry, the number of peers is equal to the number of entries.

Data Load

The number of entries per peer is one of our key performance indicators, as well-balanced data are the prerequisite for good balancing of the network

Figure 8. Entries per Peer, using n=26 (left), and n=5 (right)



load. Figure 8 shows the number of entries per peer for m in 25, 50, 75, and 100, without replication.

Note that the y-scale showing the number of peers is logarithmic. Nearly all of the 620,853 peers store less than 3 entries. No peer stores more than 150 entries. Assuming an average size of an entry of 128 Bytes, a peer holding 150 entries would store less then 19 kBytes. This is feasible even on embedded devices with a built-in peer-to-peer stack, and it is easily possible to replicate 19 kBytes through current Internet connections.

Prefix Length

In the description of the algorithm above, we said that the initial prefix length to start with when searching in an EPHT is 5 in our dataset. As shown in Figure 9, this is the average prefix length for n in 5, 13, and 26. Only binary EPHTs with n=2 result in a significantly larger average prefix length. If the average prefix length changes over time, e.g. if the number of users or the distribution of names is other than expected, then the initial prefix length needs to be adapted in the search operation.

Network Traffic

The network traffic is evaluated in terms of the number of lookup operations in the DHT that is needed to process a range query[2]. As an example, we queried the prefixes SCHN* which results in

Figure 9. Prefix Length for m=25 (left), and m=100 (right)

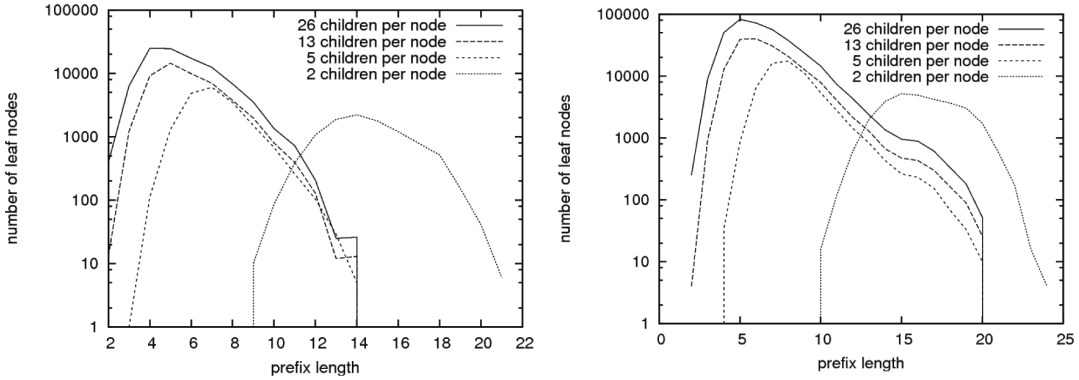


Table 1. Lookup operations

	SCHN*			OLPP*		
	N=5	*n=13*	*n=26*	*n=5*	*n=13*	*n=26*
M=50	*1068*	*958*	*495*	*99*	*3*	*4*
M=100	*590*	*670*	*279*	*39*	*3*	*4*

4683 entries, and OLPP* yielding only a single entry. Of course querying SCHN* is an artificial example, as real-world applications would probably abort that query after a certain number of results is retrieved, and ask the user to formulate the query more specifically. Table 1 shows the number of lookup operations. We did not use any caching.

An increasing maximum load of the root node m results in less nodes to be looked up. With regards to the number of children n we found that more children per node result in a lower number of lookup operations. For example, if the user searches for OLPP* in a tree with n=5, the application searches all entries matching the prefix [K-O][K-O] [P-T] [P-T]. People with a last name starting with Lost would match the same prefix as Olpp. Altogether, the number of matching entries in our phone book is 1801, which explains the overhead of 99 lookups.

These results suggest that the number of children per node *n* should be as large as possible to reduce the number of lookup operations.

Empty Nodes

The percentage of empty nodes is shown in Table 2.

As expected, the number of empty nodes raises with the number of children per node.

Table 2. Empty nodes

	n=5	*n=13*	*n=26*
m=50	*6% of 34,821*	*37% of 94,297*	*60% of 193,801*
m=100	*2% of 18,353*	*29% of 48,757*	*53% of 98,501*

However, even with *n*=26 we got only 60% empty nodes, which is still justifiable in the face of the great reduction of traffic overhead for *n*=26.

Churn

In peer-to-peer terminology, the continuous arrival and disappearance of peers is called churn. The stability of DHTs in the face of churn and the probability of data loss was addressed many times before (Stutzbach 2006, Kunzmann 2009), and we refer the reader to these works for experimental and analytical results on the topic.

The tree nodes of the EPHT are stored as resources on a DHT. DHTs use replication techniques and stabilization protocols to keep the probability of data loss very low, even in typical file-sharing scenarios where the participating peers arrive and disappear very frequently.

The reliability of the EPHT depends on the reliability of the underlying DHT. If the node resources are available on the DHT layer, then the EPHT remains stable. Assuming that VoIP telephones have much longer average online times than file sharing peers, we expect the DHT to be very stable in the distributed phone book scenario.

However, in order to handle the unlikely event of data loss, we propose that the peers look up their own entry on a periodical basis, and re-publish the entry in case it disappeared.

CONCLUSION

In this article, we presented the Extended Prefix Hash Tree as an infrastructure supporting range

queries on top of Distributed Hash Tables. The design of the algorithm is driven by the requirements found in a distributed user directory for a commercial VoIP communication platform developed by Siemens. We evaluated the algorithm and showed how to choose the parameters in order to achieve the best performance.

While this article is focused on a specific use case, the methodology and results can be transferred to other scenarios. The algorithm presented here fits specifically in situations where keywords are Zipf-distributed. In the phone book scenario, some last names are very common while other last names are very rare. The EPHT adapts perfectly to this kind of distribution.

The concatenation of keywords provides a simple but powerful approach to handle multiple keywords that are ordered in a hierarchical way.

FUTURE RESEARCH DIRECTIONS

The Extended Prefix Hash Tree algorithm presented in this paper enables the implementation of a distributed user directory for a peer-to-peer-based telephony application. However, apart from user directories, there are more applications that might benefit from a distributed search index.

The evaluation in this article is based on the specific requirements that we derived from a commercial communication platform. When EPHTs are to be applied in other applications, it is a non-trivial task to tell the implications of the algorithm on the specific architecture.

Future research should address this issue and allow for the definition of generic, re-usable components that can be applied on top of peer-to-peer networks. These components are the building blocks fulfilling the application-specific requirements on the distributed infrastructures. A first proposal for the definitions of these components can be found in (Stäber, 2009).

Also, while DHT-based structured overlay networks have many advantages, their string-based approach for registration and lookup carries intrinsic limitations as regards the expressiveness of search. While the extension with range queries and wildcard search seems appropriate for a pure phone book lokup, even a straightforward business directory will require more semantically elaborate queries (e.g., SQL-based or ontology-based queries). One option to achieve this is to combine structured distributed hash tables with super-peer architectures, preserving the robustness and scalability of the overlay while enhancing it with declarative semantic search capability. In (Gerdes et al., 2009), we propose a declarative decentralized query processor and evaluate it in the energy domain. (Stiefel and Müller, 2010) propose the use of an ontology-based query language on top of a DHT architecture for semantic search of digital product models. These approaches will need to be validated and further developed in future work.

REFERENCES

Aspnes, J., & Shah, G. (2003). Skip graphs. In *SODA '03: Proceedings of the Fourteenth Annual ACM SIAM Symposium on Discrete Algorithms*, (pp. 384–393). Philadelphia, PA, USA.

Barsanti, L., & Sodan, A. (2007). Adaptive job scheduling via predictive job resource allocation. In *Proceedings of Job Scheduling Strategies for Parallel Processing* (pp. 115-140).

Bellare, M., & Rogaway, P. (1993). *Random oracles are practical: A paradigm for designing efficient protocols*. In CCS '93 1st ACM Conference on Computer and Communications Security, (pp. 62–73). New York, NY: ACM Press.

Bharambe, A. R., Agrawal, S., & Seshan, S. (2004). *Mercury: Supporting scalable multi-attribute range queries*. In SIGCOMM Symposium on Communications Architectures and Protocols, (pp 353–366). Portland, OR, USA.

Buyya, R., Giddy, J., & Abramson, D. (2000). *An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications.* Paper presented at the Second Workshop on Active Middleware Services (AMS2000), Pittsburgh, USA.

Gerdes, C., Eger, K., & Müller, J. P. (2009). *Data-centric peer-to-peer communication in power grids.* Electronic Communications of the EASST 17: Kommunikation in Verteilten Systemen 2009, 2009. *Proceedings of KiVS Global Sensor Networks Workshop* (GSN09).

Hautakorpi, J., & Schultz, G. (2010). *A feasibility study of an arbitrary search in structured peer-to-peer networks.* In ICCCN'10: *Proceedings of the 19th International Conference on Computer Communications and Networks*. Zurich.

Indyk, P., Motwani, R., Raghavan, P., & Vempala, S. (1997). Locality-preserving hashing in multidimensional spaces. In STOC '97: *Proc. of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, (pp. 618–625). New York, NY: ACM Press.

Liu, L., & Lee, K.-W. (2004). Supporting efficient keyword-based file search in peer-to-peer file sharing systems. In GLOBECOM'04: *Proc. of the IEEE Global Telecommunications Conference*.

Ramabhadran, S., Ratnasamy, S., Hellerstein, J. M., & Shenker, S. (2004). *Prefix hash tree – an indexing data structure over distributed hash tables*. In PODC'04*:* 23rd Annual ACM Symposium on Principles of Distributed Computing.

Risson, J., & Moors, T. (2006). Survey and research towards robust peer-to-peer networks: Search methods. *Computer Networks*, *50*(17), 3485–3521. doi:10.1016/j.comnet.2006.02.001

Schmidt, C., & Parashar, M. (2004). Enabling flexible queries with guarantees in P2P systems. *IEEE Internet Computing*, *8*(3), 19–26. doi:10.1109/MIC.2004.1297269

Stäber, F. (2009). *Service layer components for decentralized applications*. Doctoral Dissertation at the Clausthal University of Technology

Stiefel, P. D., & Müller, J. P. (2010). A model-based software architecture to support decentral product development processes. In: *Exploring the grand challenges for next generation e-business. Proceedings of the 8th Workshop on eBusiness* (Web 2009). *Volume 52 of Lecture Notes in Business Information Processing*. Springer-Verlag, 2010. To appear.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In SIGCOMM'01: *Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, (pp. 149–160). San Diego, CA: ACM Press.

Stutzbach, D., & Rejaie, R. (2006). Understanding churn in peer-to-peer networks. In IMC'06: *Proc. of the 6th ACM SIGCOMM on Internet Measurement*, (pp. 189–202). New York, NY: ACM Press.

ADDITIONAL READING

Aberer, K., & Hauswirth, M. (2004). *Peer-to-Peer Systems, Practical Handbook of Internet Computing*. Baton Rouge: Chapman Hall & CRC Press.

Baset, S. A., & Schulzrinne, H. (2004). *An analysis of the Skype Peer-to-Peer Internet telephony protocol. Tech. report*. New York, USA: Columbia University.

Binzenhöfer, A., Staehle, D., & Henjes, R. (2005): *On the stability of Chord-based P2P systems*. In GLOBECOM '05: Proc. of the IEEE Global Telecommunications Conference.

Binzenhöfer, A., & Tran-Gia, P. (2004): *Delay analysis of a Chord-based peer-to-peer file-sharing system*. In ATNAC '04: Proc. of the Australian Telecommunication Networks and Applications Conference.

Biondi, P. and Desclaux F. (2006): *Silver needle in the Skype*. Black Hat Europe 2006.

Dabek, F., Zhao, B., Druschel, P., & Kuiatowicz, J. (2003): *Towards a common API for structured peer-to-peer overlays.* In IPTPS'03: Peer-t-Peer Systems II, Second International Workshop, volume 2734 of Lecture Notes in Computer Science, pages 33—34, Berlin, Heidelberg: Germany, Springer

Eberspaecher, J., & Schollmeier, R. (2005): *Peer-to-Peer systems and applications*. chapter First and Second Generation of Peer-to-Peer Systems, pages 35—56, Springer.

Eyers, T., & Schulzrinne, H. (2000): *Predicting Internet telephony call setup delay.* In IPTel 2000: Proc. of the 1st IP-Telephony Workshop.

Friese, T., Freisleben, B., Rusitschka, S., & Southall, A. (2002): *A framework for resource management in peer-to-peer networks.* Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World, Lecture Notes In Computer Science, volume 2591, Springer, 2002, pages 4—21.

Friese, T., Müller, J. P., & Freisleben, B. (2005). *Self-Healing Execution of Business Processes Based on a Peer-to-Peer Service Architecture.* In: Proc. 18th Int. Conference on Architecture of Computing Systems [Springer.]. *Lecture Notes in Computer Science*, *3432*, 108–123. doi:10.1007/978-3-540-31967-2_8

Ganesan, P., Yang, B., & Garcia-Molina, H. (2004): *One torus to rule them all: multi-dimensional queries in P2P systems*. In WebDB '04: Proc. of the 7th International Workshop on the Web and Databases, ACM Press, pages 19—24.

Garcés-Erice, L., Felber, P. A., Biersack, E. W., Urvoy-Keller, G., & Ross, K. W. (2004): *Data indexing in peer-to-peer DHT networks*. In ICDCS '04: Proc. of the 24th International Conference on Distributed Computing Systems, IEEE Computer Society, pages 200—208.

Guha, S., Daswani, N., & Jain, R. (2006): *An experimental study of the Skype peer-to-peer VoIP system*. In IPTPS '06: Proc. of the 5th International Workshop on Peer-to-Peer Systems.

Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., & Stoica, I. (2003). The impact of DHT routing geometry on resilience and proximity. In *SIGCOMM '03: Proc. of the conference on Applications, technologies, architectures, and protocols for computer communications* (pp. 381–394). ACM Press. doi:10.1145/863955.863998

Kellerer, W., Kunzmann, G., Schollmeier, R., & Zoels, S. (2006). Structured peer-to-peer systems for telecommunications and mobile environments. *AEÜ. International Journal of Electronics and Communications*, *60*(1), 25–29. doi:10.1016/j.aeue.2005.10.005

Kunzmann, G. (2009): *Performance Analysis and Optimized Operation of Structured Overlay Networks.* Doctoral thesis, Technische Universitaet Muenchen.

Kunzmann, G. and Binzenhoefer A. and Stäber, F. (2008): *Structured overlaynetworks as an enabler for future internet services*. it - Information Technology volume 50, no. 6, pages 376—382.

Lennox, J., & Schulzrinne, H. (2000). *Feature interaction in Internet telephony. Feature Interactions in Telecommunications and Software Systems VI* (pp. 38–50). IOS Press.

Leslie M. and Davies J. and Huffman. (2006): *Replication strategies for reliable decentralized storage.* In ARES'06: Proc. of the First International Conference on Availability, Reliability and Security. pages 740—747.

Li, J., Stribling, J., Morris, R., Kaashoek, M. F., & Gil, T. M. (2005): *A performance vs. cost framework for evaluating DHT design tradeoffs under churn*, In INFOCOM '05: Proc. of the 24th Joint Conference of the IEEE Computer and Communications Societies, pages 225—236.

Liu, L., & Lee, K.-W. (2004): *Keyword fusion to support efficient keyword-based search in peer-to-peer file sharing.* In CCGRID'04: Proc. of the 2004 IEEE International Symposium on Cluster Computing and the Grid, IEEE Computer Society, pages 269—276.

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., & Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, *7*(2), 72–93. doi:10.1109/COMST.2005.1610546

Maymounkov, P., & Mazières, D. (2006): *Kademlia: A peer-to-peer information system based on the xor metric.* In IPTPS'02: Proc of the 1st International Workshop on Peer-to-Peer Systems, pages 53—65, London: UK, Springer

Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., et al. (2002): *Peer-to-peer computing*. Technical Report HPL-2002-57, HP Labs, Palo Alto, CA, USA.

Oram, A. (Ed.). (2001). *Peer-to-Peer, Harnessing the Power of Disruptive Technologies*. Sebastopol, CA, USA: O'Reilly.

Ratnasamz, S., Francis, P., Handley, M., Karp, R., & Schenker, S. (2001): A scalable content-addressable network. In SIGCOMM'01: Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pages 161—172, New York, NY: USA, ACM Press.

Reynolds, P., & Vahdat, A. (2003): Efficient peer-to-peer keyword searching, Proceedings of International Middleware Conference, Lecture Notes in Computer Science, vol. 2672, Springer, pages 21—40.

Rhea, S., Geels, D., Roscoe, T., & Kubiatowicz, J. (2003): *Handling churn in a DHT*. Tech. Report UCB/CSD-03-1299, EECS Department, University of California, Berkeley.

Rowstron, A., & Druschel, P. (2001): *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.* In Middleware'01: Proc. of the IFIP/ACM International Conference on Distributed Systems, volume 2218 of Lecture Notes in Computer Science, pages 329—350.

Rusitschka, S., & Southall, A. (2003). *The resource management framework: A system for managing metadata in decentralized networks using peer-to-peer technology.* In Agents and Peer-to-Peer Computing. In *Lecture Notes in Computer Science* (*Vol. 2530*, pp. 144–149). Springer.

Sarma, A., Bettstetter, C., Dixit, S., Kunzmann, G., Schollmeier, R., & Nielsen, J. (2006). *Self-organization in communication networks* (pp. 423–451). Wiley.

Seedorf, J. (2006). Security challenges for Peer-to-Peer SIP. *IEEE Network*, *20*, 38–45. doi:10.1109/MNET.2006.1705882

Shu, Y., Ooi, B. C., Tan, K.-L., & Zhou, A. (2005): Supporting multi-dimensional range queries in peer-to-peer systems, In P2P '05: Proc. of the 5th IEEE International Conference on Peer-to-Peer Computing, IEEE Computer Society, pages 173—180.

Sit, E., & Morris, R. (2002): *Security considerations for peer-to-peer distributed hash tables.* In IPTPS '02: Proc. of the 1st International Workshop on Peer-to-Peer Systems.

Spleiss, C., & Kunzmann, G. (2007). *Decentralized supplementary services for Voice-over-IP telephony.* Proceedings of EUNICE 2007. *Lecture Notes in Computer Science*, *4606*(Jul), 62–69. doi:10.1007/978-3-540-73530-4_8

Steinmetz, R., & Wehrle, K. (2005): *What is peer-to-peer about?* In volume 3485 of Lecture Notes in Computer Science, pages 9—16. Berlin, Heidelberg: Germany, Springer.

Stutzbach, D., & Rejaie, R. (2006): *Understanding churn in peer-to-peer networks*. In IMC'06: Proc. of the 6th ACM SIGCOMM on Internet Measurement, pages 189—202, New York, NY, USA: ACM Press.

Tanenbaum, A. (2003). *Computer Networks*. Upper Saddle River, NJ, USA: Prentice Hall International.

Tutschku, K., & Tran-Gia, P. (2005). *Peer-to-peer-systems and applications. chapter Traffic Characteristics and Performance Evaluation of Peer-to-Peer Systems* (pp. 383–397). Springer.

Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A., & Kubiatovicz, J. (2004). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, *22*(1), 41–53. doi:10.1109/JSAC.2003.818784

Zoels, S., Schubert, S., & Kellerer, W. (2006): *Hybrid dht design for mobile environments*. In AP2PC'06: Proc of 5th International Workshop on Agents and Peer-to-Peer Computing, LNCS, Springer.

Zuo, C., Li, R., Shen, H., & Lu, Z. (2009): High Coverage Search in Multi-Tree Based P2P Overlay Network. In ICCCN'09: Proc of the 18th International Conference on Computer Communications and Networks. San Francisco, CA: USA.

KEY TERMS AND DEFINITIONS

Decentralization: Attempt to avoid central services, thus preventing single points of failure.

Directory: A service organizing users

Distributed Hash Table (DHT): Type of decentralized infrastructure providing a hash-table like addressing scheme

Extended Prefix Hash Tree (EPHT): Modified PHT to be used when implementing distributed user directories.

Infrastructure: Underlying algorithm in a distributed application, allowing the nodes to address each other.

IP-Telephony: Telephony over IP networks, mostly using the Voice over IP protocol

Peer-to-Peer: A paradigm in distributed systems, where all nodes may act as both, client and server.

Prefix Hash Tree (PHT): Search algorithm based on DHTs, as proposed by Ramabhadran et al (2004)

ENDNOTES

1 Dr. G. Kunzmann is now working for DOCOMO Communications Laboratories Europe GmbH, Munich, Germany

2 Each lookup operation requires $\log_b n$ messages in the DHT protocol, where n is the number of peers, and b is a parameter depending on the design of the DHT's routing table.