Fast Similarity Search in Peer-to-Peer Networks

Thomas Bocek[1], Ela Hunt[2], David Hausheer[1], Burkhard Stiller[1,3]
[1]Communication Systems Group, Department of Informatics IFI, University of Zurich, Switzerland
[2]Department of Computer Science, ETH Zürich, Switzerland
[3]Computer Engineering and Networks Laboratory TIK, ETH Zürich, Switzerland

E-mail: [bocek|hausheer|stiller]@ifi.uzh.ch, hunt@inf.ethz.ch

***Abstract*—Peer-to-peer (P2P) systems show numerous advantages over centralized systems, such as load balancing, scalability, and fault tolerance, and they require certain functionality, such as search, repair, and message and data transfer. In particular, structured P2P networks perform an exact search in logarithmic time proportional to the number of peers. However, keyword similarity search in a structured P2P network remains a challenge. Similarity search for service discovery can significantly improve service management in a distributed environment. As services are often described informally in text form, keyword similarity search can find the required services or data items more reliably.**

This paper presents a fast similarity search algorithm for structured P2P systems. The new algorithm, called P2P Fast Similarity Search (P2PFastSS), finds similar keys in any distributed hash table (DHT) using the edit distance metric, and is independent of the underlying P2P routing algorithm. Performance analysis shows that P2PFastSS carries out a similarity search in time proportional to the logarithm of the number of peers. Simulations on PlanetLab confirm these results and show that a similarity search with 34,000 peers performs in less than three seconds on average. Thus, P2PFastSS is suitable for similarity search in large-scale network infrastructures, such as service description matching in service discovery or searching for similar terms in P2P storage networks.

***Index Terms*—P2P, DHT, similarity search, service discovery**

I. Introduction

Peer-to-peer (P2P) systems have evolved from unstructured systems like Gnutella to structured systems like Chord [21], CAN [17], Pastry [18], or Kademlia [15], which form the basis for distributed hash tables (DHTs). P2P systems can be used to overcome limitations present in centralized systems. Such limitations include low scalability, weak fault tolerance, and poor load balancing. With respect to the service discovery application domain, a key limitation is the low efficiency of service discovery procedures for text-based service descriptions. New algorithms and protocols in a P2P settings have to be designed to provide functionality normally found in centralized systems.

The most basic operations in a DHT are *get(key)* and *put(key, value)*. Data is stored in a DHT by applying a hash function to the key and calling the put method. A search is performed using the get method. The same hash function is applied to the query. The benefit of using a structured P2P system is that a key lookup typically requires *O(log n)* routing steps, where *n* is the number of peers in the system. However, DHTs have limited support for similarity search, because calculating a hash function on similar keys results in different hash values. DHT-based systems lack support for similarity search using the edit distance metric. Existing solutions are limited to n-gram based substring searches [2] and IR-type ranking based on the Jaccard coefficient [5].

The main contribution of this paper is a content-based full-text fast similarity search algorithm, called P2PFastSS, which uses the edit distance metric and is applicable to the service discovery domain. The edit distance [13] is the minimum number of operations required to transform one string into another, using the operations: deletion, insertion, and replacement. The proposed algorithm can be used on top of any DHT, since it uses only the basic operations get and put. Thus, P2PFastSS enables a peer to search for a similar keyword in any text-based content, such as texts, service descriptions, or abstracts. In the following, the term document refers to a service or data description. It is assumed that these documents are stored randomly on peers and are indexed either by the peer storing the document or by the peer sending the document for storage. This holds especially true for a distributed service discovery environment in which multiple decentralized services are offered independently to various service users.

P2PFastSS returns documents that contain similar keywords, with ranking based on the edit distance. This means that documents, such as textual service or data descriptions containing an exact occurrence of a word, are returned first, then the documents containing the keyword with one insert, delete or replacement, and then other documents, ordered by the edit distance of the search term and words in the document. Such searches can be applied to a variety of documents, where the spelling of the query term is not known, or may have been changed due to poor transcription or multilingual context.

A potential example of use is, as mentioned, service discovery based on service descriptions. Beside using a centralized architecture, UDDI [23], RMI [11], or Corba [8] use a method name lookup for service discovery. However, if this method name is only known approximately or if it is misspelled, the

service is not found. The key benefit of using P2PFastSS is that services can also be found by searching in service descriptions, *e.g.,* in the *documentation* tag defined in WSDL [7], even in the case of misspelled search queries.

With respect to related work in similarity search, several recent papers propose to change the routing scheme or to build a new topology in order to support similarity searches, see [3], [4], and [22]. In contrast, P2PFastSS does not need to change the topology or the routing to search for similar keys using the edit distance metric. Therefore, P2PFastSS can be applied to any DHT.

Performance analysis and experimental evaluations show that the approximate keyword search P2PFastSS, which is based on text indexing, requires *O(log n)* time and communication messages where *n* is the number of peers, while the indexing of a word requires *O(log n)* time and communication messages. Time measurements on PlanetLab show that both the similarity indexing and similarity search perform in less than three seconds on average.

The remainder of the paper is structured as follows. Section 2 presents related work and Section 3 introduces P2PFastSS. While Section 4 provides the theoretical analysis, Section 5 shows the implementation and experimental evaluation. Section 6 presents a discussion and conclusion.

II. Related Work

A naive approach to similarity searching is to use broadcasting or flooding based P2P networks. Each peer in the network will receive the query and can locally execute a pattern matching algorithm. However, broadcasting or flooding typically do not scale well. This section describes the algorithms related to similarity search and related work that addresses scalability issues.

A. Similarity Search Algorithms

The edit distance [13] is the minimum number of operations required to transform one string into another. An operation is either an insertion, a deletion, or a replacement. The edit distance is calculated using dynamic programming (DP) in *O(pq)* time where *p* and *q* are the lengths of strings being compared. For example, the edit distance between *house* and *mouse* is 1 because the minimum operation to transform *house* into *mouse* is to replace *h* with *m*.

N-grams [14] are created by sliding a window of length *n* over the data and storing the content and position of all windows. N-grams can be used in similarity searching based on matching grams. A high number of matching grams suggests high similarity. It is possible to use the edit distance metric with n-grams. However, if the edit distance is large and the word length is short, n-grams fail to efficiently find similar data. N-grams can be used with a global metric, such as cosine similarity, which finds similar terms in a very efficient manner. However, as the metric is global in the cosine similarity, the edit distance metric cannot be applied.

FastSS [6] performs a similarity search using the edit distance metric and is faster than DP. However, the data has to be indexed beforehand. Given a constant word length and edit distance, FastSS finds similar words in *O(log d)*, where *d* is the dictionary size. FastSS uses a deletion neighborhood to lookup similar data. In contrast to n-grams, FastSS performs well if the word length is small. Fig. 1 shows an example of FastSS. The deletion neighborhood of the right hand side word (*fest*) has been indexed beforehand, while on the left hand side the deletion neighborhood of *test*, which is the query, is shown. The algorithm will return an edit distance of 1 as the neighbor *est* is shared.

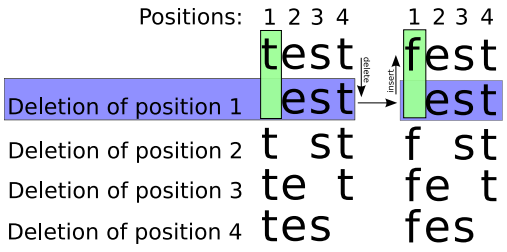

Fig. 1. FastSS example, transformation from test to fest

B. P2P Algorithms: Distributed Hash Table

Distributed hash tables (DHTs) as in Chord [21], CAN [17], Pastry [18], and Kademlia [15] introduce structures into a P2P network. In a structured network, a query is not forwarded to all peers as in a broadcasting or flooding based P2P system, but to a selected set of peers. This selection process uses an overlay structure which defines a closeness metric. A DHT provides the operations *get(key)* and *put(key,value)*, and typically performs in *O(log n)* routing steps. However, DHTs do not support exhaustive similarity searching.

C. P2P Similarity Search Algorithms

Table I summarizes the related work with regard to the following three dimensions: structuring, search paradigm, and the type of indexing. EK stands for exact keyword, TK for top-k (or k nearest neighbor), RQ for range queries, SUB for substring, AK for approximate keyword, IR for information retrieval based on variants of the Jaccard coefficient, and PT for prefix tree.

TABLE I
Related Work Categorization

	Structure	**Search**	**Indexing**
Gnutella 0.6 [10]	Ultra peers	EK on several terms	none
Pier [9]	DHT	EK, RQ	PT
P-Grid [1], AlvisP2P [20]	DHT	EK, SUB, RQ, IR	PT
LSH forest [5]	DHT (P-Grid)	IR	PT
pSearch [22]	DHT (CAN)	IR	Rolling index
Aekaterinidis et al. [2]	DHT	EK, SUB, AK	None/n-grams
SWAM [4]	SWAM-V	EK, RQ, TK	SWAM-V

TABLE I
RELATED WORK CATEGORIZATION

	Structure	Search	Indexing
Karnstedt et al. [12]	DHT	EK, SUB, RQ, AK	N-grams, vertical partition
Litwin et al. [14]	none	EK	none
Batko et al. [19]	DHT	Metric space search	VPT, GHT, MCAN, M-Chord
Ahmed et al. [3]	Lattice Pattern matching	Pattern matching, AK, SUB	Bloom filter
P2PFastSS	DHT	AK	Deletion neighborhood

Gnutella 0.6 [10] is a file sharing system which uses a hierarchy of leaves and ultra peers who hold knowledge required for searching on sets of keywords. It is known to be unable to find rare data items, and other approaches like Pier [9] remedy this by combining DHT with prefix trees which index keys of varying length and can support range queries by either walking along the leaves in a range, using leaf pointers, or by first finding a prefix shared by a range and then finding all children reachable via the prefix. A special path query primitive is used by Pier to locate nodes based on a prefix string. Similarly, P-Grid [1] uses prefix trees and, later, AlvisP2P [20] offers full IR functionality, based on selected posting lists. LSH Forest [5] uses P-Grid and focuses on optimizing the performance of text indexing by developing further the locally-sensitive hashing technique, particularly for skewed data distributions. The actual solution includes using several overlays with various hash function combinations, treated as a forest. pSearch [22] uses latent semantic indexing to cluster nodes in the network and deliver better IR searching via a better locality of similar answers.

Beside IR type matching, keyword, range, top-k, and substring matching are relevant in P2P. Recent work by Aekaterinidis and Triantafillou [2] extends their previous solutions and examines the use of PastryStrings and DHTs in substring matching, based on a subdivision of a query into all possible substrings and querying for those, or, alternatively, querying for n-grams which are substrings of length *n* which can be indexed in a DHT. N-grams lead to lots of network traffic while query splitting methods offer tradeoffs between space and traffic, with no method being a clear winner. SWAM [4] focuses on EK, RQ and TK queries, and uses a Voronoi-diagram based graph network overlay (SWAM-V) to express similarity between items held in the network. It appears that in networks with high churn SWAM-V might not be as effective as in more static scenarios. Karnstedt et al. [12] address the issue of complex queries (EK, SUB, RQ) in the P-Grid based DHT. They split relational tables vertically, to support richer predicates including attribute values, and index those in a DHT. They also use an n-gram index to provide approximate string search. The authors find that n-grams are appropriate for searching in long descriptions, but not in short words, which results from the fact that to assess similarity, one compares the number of shared n-grams between the query and target, and takes into account both target and query lengths. Thus, for short strings and short targets this approach is not effective. Litwin and colleagues [14] offer a new string searching approach which uses no indexing, but encodes strings using algebraic signatures on n-grams. This ensures data privacy, but still incurs a large cost in string searching, which is faster than other approaches, but will be slower than indexing. Batko et al. [19] analyze the performance of multidimensional indexing in P2P. As this type of indexing incurs cost linear in the size of the data set, distributed search offers performance gains via parallelization. They test the performance of search in several scenarios, and conclude that none of the methods they tested is a clear winner. Ahmed et al. [3] describe a distributed pattern matching technique. They support flexible queries such as partial keywords and multiple keywords. The routing algorithm is based on Bloom filters. The search speed is comparable to a DHT and its time is logarithmic in the number of peers.

D. Comparison with other Approaches

Apart from P2PFastSS, a P2P search based on the edit distance can be built using concepts presented in Ahmed et al., Karnstedt et al., and Aekaterinidis et al. However, the later two approaches do not work efficiently with small words in distributed systems, as pointed out by [12] and studied by [6] in centralized systems. The first approach has defined its own routing based on Bloom filters. The advantage of P2PFastSS is the independence of the routing algorithm, as P2PFastSS can use any DHT.

III. DESIGN OF P2PFASTSS

P2PFastSS finds text documents containing keywords similar to the ones included in the search query. Similarity in P2PFastSS is defined as the edit distance *k* between keywords. In the following sections the term *node* is used in the context of P2PFastSS synonymously with the term *peer*.

The key requirements of the design are as follows.
(A) The insertion of a keyword location tuple in *O(log n)* time and communication messages, where *n* is the number of nodes.
(B) The removal of a keyword location tuple in *O(log n)* time and communication messages.
(C) The retrieval of similar keywords under the edit distance metric in *O(log n)* time and communication messages.

P2PFastSS is built on top of a structured P2P network in order to achieve a lookup time logarithmic in the number of nodes. The algorithm is split into two phases. In the first phase the document is stored and indexed, while in the second phase the similarity search is performed.

A. Indexing and Storing (First Phase)

In the first phase documents in P2PFastSS are stored using the DHT operation *put(key, value)*. The key of the document is the hash of the document title. The value of such an entry is the document itself (see Table II).

All words included in the document need to be indexed before a similarity search can be performed. Indexing is performed using the following steps.

1) All words in the document are identified.

2) A deletion neighborhood is generated for all words in the document.

3) All neighbors are stored with a reference to the document IDs in which the word appears, using the operation *put(key, value)*. As words may appear in many documents, a key can hold multiple values (a list of references).

The method P2PFastSS_index in Fig. 2 shows this indexing procedure in pseudo code. The deletion neighborhood is generated by the method precalculate which takes the edit distance k as an argument..

```
P2PFastSS_index(String text) {
  StringList words=split(text)
  DHT.put(genkey(text),text)
  StringList pVariants = precalculate(words,k)
  for pMatch in (pVariants) {
    DHT.put(getkey(pMatch),(pMatch,genkey(text))
  }
}
```

Fig. 2. P2PFastSS pseudo code for indexing

In the following example, keys and values of index entries are shown. Table II shows the keys and values of a node's document table. Table III shows keys and values of a node's keyword index, and Table IV neighborhood entries for *test* with *k=1*. The key is determined by the hash of the document title (Table II) or keyword (Table III). The value contains the text (Table II), or the keyword and a reference that points to the document (Table III). If a word appears in several documents, it is mapped to a list of locations.

TABLE II
DOCUMENT INDEX EXCERPT OF NODE 0X1235

Title (Key)	Document (Text)
Albedo (Doc ID: 0x123)	The albedo of an object ...
Achilles (Doc ID: 0x132)	In Greek mythology, Achilles ...
Paper (Doc ID: 0x238)	Paper is a commodity of thin ...
Testing (Doc ID: 0x321)	This test aims to ...

During the indexing phase P2PFastSS generates a deletion neighborhood by using the method precalculate (see Fig. 2) and will index all neighbors of word *test* (Table IV). The messages sent by the searching node in the process of indexing and storing a neighbor in a distributed environment are shown in Fig. 3. Node 0x1 wants to index the deletion neighborhood of the word *test* (hash 0x563), which points to the document with ID 0x321. In messages 1 and 2, it finds the locations appropriate for placing the index entries for the first neighbor *tes*, which involves two routing queries. Then *tes* is stored redundantly in message 3. Messages 1 and 2 are sent sequentially, while messages 3 can be sent in parallel. This process continues until all neighbors and the keyword are stored.

TABLE III
KEYWORD INDEX EXCERPT OF NODE 0X1235

Keyword (Key)	Resource and Location
object (hash 0x424)	object, Doc ID: 0x123...
pper (hash 0x927)	paper, Doc ID: 0x238... piper, Doc ID: 0x641...
wter (hash 0x149)	water, Doc ID: 0x583...
tes (hash 0x123)	test, Doc ID: 0x321...

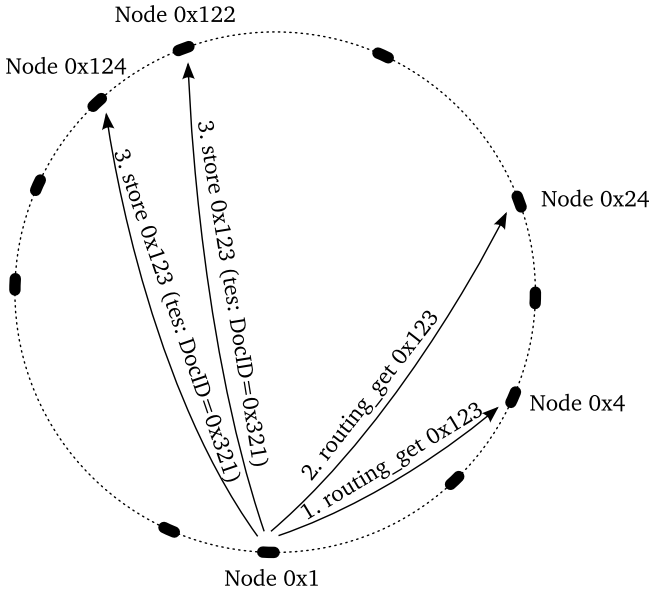

Fig. 3. Message diagram of the indexing phase of tes (hash:0x123)

TABLE IV
DELETION NEIGHBORHOOD OF TEST, K=1

Neighbors	Resource Location
tes	(hash: 0x123, DocID: 0x321)
tet	(hash: 0x374, DocID: 0x321)
tst	(hash: 0x967, DocID: 0x321)
est	(hash: 0x368, DocID: 0x321)

B. Searching (Second Phase)

The similarity search is performed in the second phase using the following steps.

1) A node generates a deletion neighborhood from the search keyword.

2) Every neighbor is searched for using *get(key)*. The result contains a document ID and the keyword. For a given *k*,

neighbors with an edit distance *k* or larger (up to *2k)* are found [6]. Before the document is retrieved using *get(key)*, potential matches are confirmed using DP (see Section II) to check if they do not exceed the given edit distance (cf. Fig. 4). This algorithm is known as FastSSwC [6], developed by these authors.

3) The document is retrieved with *get(key)* using the matching document IDs found in step 2.

```
P2PFastSS_search(String keyword) {
  StringList result = new StringList()
  StringList pVariants = precalculate(p,k)
  for pMatch in (pVariants) {
    candidate,key=DHT.get(pMatch)
    if(DP (candidate, keyword) <= 1) {
      result.add(DHT.get(key))
    }
  }
}
```

Fig. 4. P2PFastSS pseudo code for searching

The message diagram for the search in a distributed environment is shown in Fig. 5. Node 0x1 searches for the word *tesa*. Before searching, P2PFastSS generates a deletion neighborhood from *tesa*: (*tes,tea,tsa,esa*) and starts by searching for the first neighbor *tes*.

The key for *tes* is 0x123. First, the node address is found via messages 1 and 2. In message 3, the node containing *tes* is looked up. Messages 1, 2, and 3 are sent sequentially. Finally, node 0x122 returns the Doc ID, which can then be used to retrieve the document containing the term *tes* which is similar to *tesa* (edit distance 1).

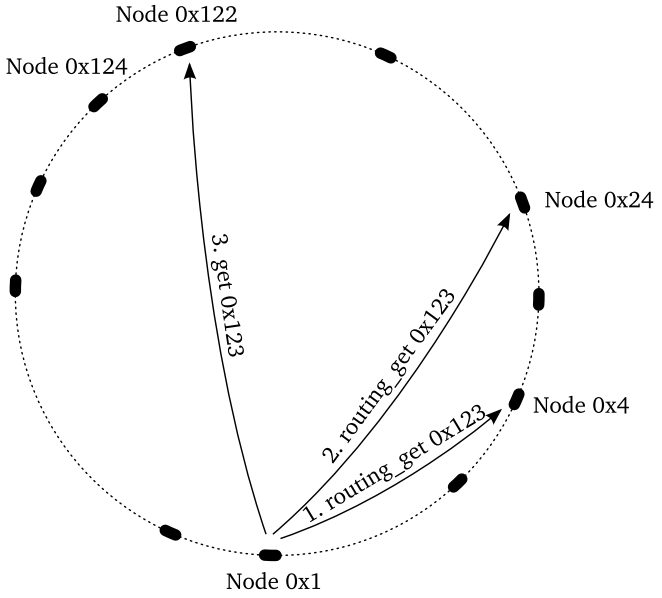

Fig. 5. Message diagram of the searching phase of tes (hash:0x123)

IV. Theoretical Analysis

The complexity of the new algorithm remains within highly practical limits, which makes it applicable in the service discovery domain with full text service descriptions. The complexity of P2PFastSS is analyzed below, with respect to the indexing, storing, and search phases. As shown below, the complexity of P2PFastSS satisfies the requirements (A), (B), and (C) defined in Section III.

A key driver for the complexity is the neighborhood generation. Neighborhood generation based on deletions has complexity $O(m^k)$, where m is the length of a word and k is the edit distance [6]. As P2PFastSS uses a deletion neighborhood, a factor of m^k is considered during searching and indexing. As P2PFastSS is based on a DHT, the complexity of the DHT operations *get* and *put* are typically *O(log n)*, where n is the number of nodes in the network.

A. Indexing and Storing Complexity

The index is distributed in the DHT. Let w be the number of words in a document, and each word be stored in *O(log n)* time, using the *put(key, value)* method. Storing all neighbors requires m^k put operations. Adding redundancy r, the storage time and communication message complexity per word is $O(r(m^k)\log n)$. Thus, for all words in a document the storage time and communication message complexity is $O(wr(m^k)\log n)$. With constant r, m, and k, indexing is performed in *O(log n)* for each word w. Thus, requirement (A), to insert a keyword location tuple in *O(log n)* time, has been met. As the remove operation can be seen as an insert storing a null value, requirement (B), to remove a keyword location tuple in *O(log n)* time, has been met as well.

B. Search Complexity

Since the similarity search generates a deletion neighborhood, $O(m^k)$ neighbors have to found in the DHT. As exact search takes *O(log n)* time and communication messages in a DHT, similarity search time and communication message complexity is $O(m^k \log n)$. So, for constant m and k, the similarity search is performed in time proportional to the logarithm of the number of nodes. Therefore, the requirement (C), to retrieve all similar keywords in *O(log n)* time, has been met.

C. Complexity Discussion

Typically, in distributed systems operations are performed in parallel. If indexing executes $w \cdot r \cdot m^k$ operations in parallel and the similarity search executes m^k operations in parallel, the similarity search and exact search perform similarly.

V. Implementation and Experimental evaluation

The implementation of P2PFastSS described below shows that the theoretical analysis presented above is a strong predictor of the experimental results produced in a distributed environment. P2PFastSS was implemented in Java and uses a DHT based on the Kademlia routing algorithm. The simulation was run with Java HotSpot(TM) Client VM (build 1.6.0_02-b05). The tests were run on approximately 360 PlanetLab hosts with a controller host located at the University of Zurich (UZH). The controller host has an Intel(R) Pentium(R) 4 CPU 3.60 GHz with 1 GB RAM.

In the experimental evaluation the edit distance k is set to 1, as a higher value of k would result in a much higher overhead. For example, if k=2, the overhead would increase by a factor

of 3.625 [6]. For language-based texts, *m* is constant. Words in this experiment are defined by the regular expression [a-zA-Z_0-9]{3,16}, which means that a word has a length between 3 and 16 characters. Such words may be textual descriptions of a service attribute. The message time-out, necessary for determining the maximal waiting time for message replies in a distributed environment, was set to 4 seconds.

The implementation has been tested and validated with the setup shown in Fig. 6. The controller host (UZH) runs a single bootstrap node which handles the bootstrapping of all other nodes, performs the tests, and collects measurements.

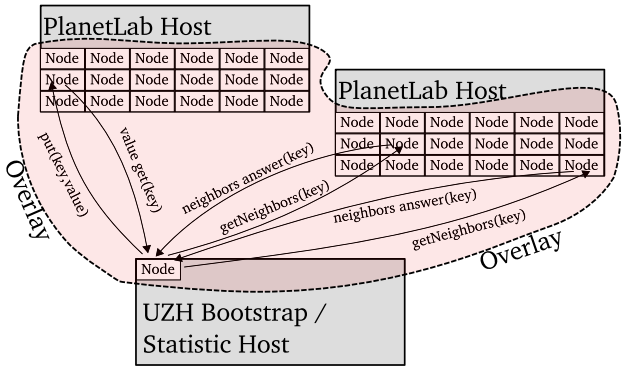

Fig. 6. Test-bed setup

Each PlanetLab host included up to 100 nodes, running in parallel and totalling in up to 34,200 nodes. The architecture of a single node is shown in Fig. 7.

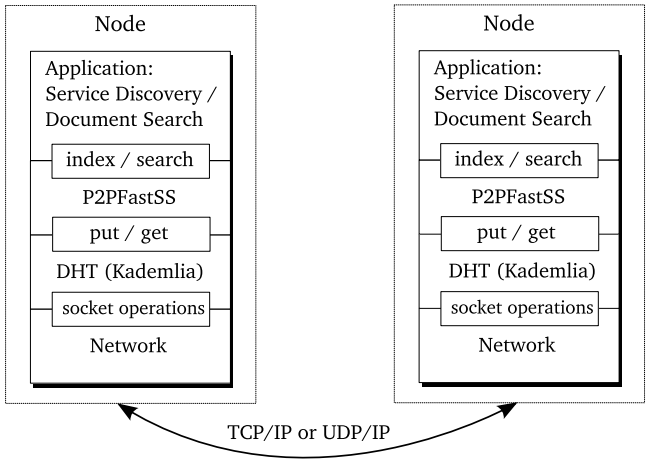

Fig. 7. Node architecture in a distributed environment

In the experiments, 100 Wikipedia abstracts were indexed. These abstracts contain in total 2,392 words with an average word length of 7 characters. Some of these words appear multiple times, such as "*the*", while other words are unique. These experiments produced measurements, categorized as message, time, and storage.

A. Message Measurements

All messages exchanged (see example in Fig. 3) were counted for P2PFastSS similarity search *and* for exact search for a keyword with an average word length. All searches were carried out 50 times and an average was taken. As shown in the theoretical analysis, the overhead introduced by P2PFastSS is m^k, which for the average word length $m=7$ and $k=1$ produces the expected factor of 7. Fig. 8 shows the mean number of messages and standard deviation for a keyword and confirms the overhead factor of 7. In addition, the graph clearly shows the logarithmic behavior of search as the number of nodes grows.

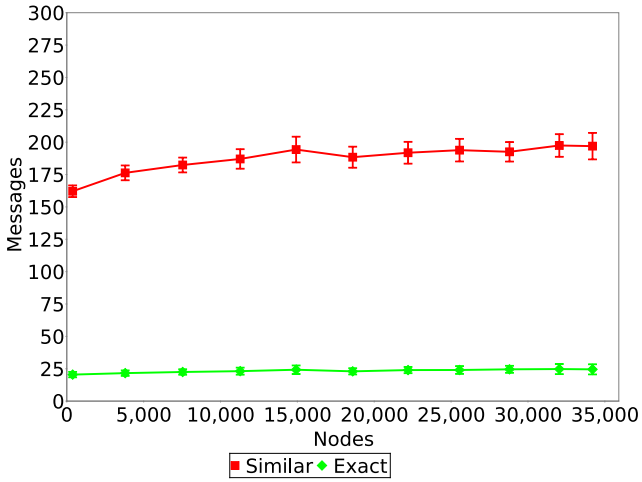

Fig. 8. Mean number of messages for a similarity and exact search, word length 7

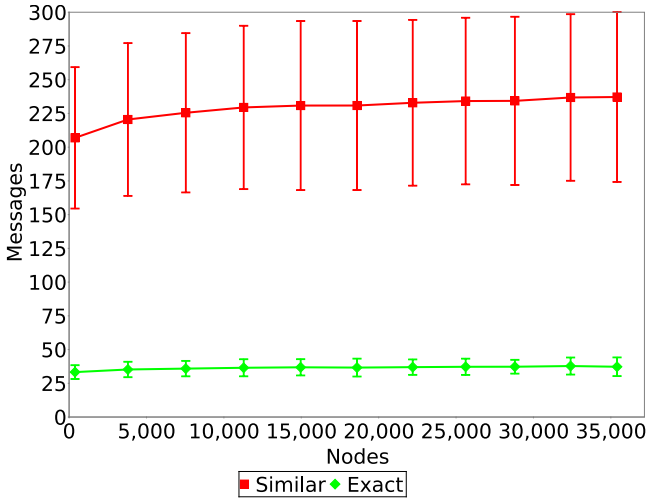

Fig. 9. Mean number of messages per keyword for similarity and exact indexing

All messages exchanged in the indexing phase were counted for P2PFastSS similarity indexing *and* exact indexing. Fig. 9 shows the mean number of messages and standard deviation for indexing a keyword. As all words were indexed, *m* is between 3 and 16, which explains the high value for the standard deviation, as short words require fewer messages. The average is based on 2,392 indexed keywords. With a lower redundancy factor for similarity indexing ($r=4$), and $r=5$ for exact search, the overhead here is lower. For a word of length $m=7$, the overhead is 5.6 (7 times 4/5). This graph con-

firms the relationship between the predicted and observed message levels.

B. Time Measurements

As mentioned in the complexity discussion, in a perfect system using parallel execution, the exact search and similarity search perform similarly with respect to the search time. However, in PlanetLab this is not exactly the case, since real-world factors, such as the current load on the host, availability, and latency, have to be considered. Fig. 10 shows execution times and the standard deviation for similarity search and exact search. Lookup times lie between 0.55 and 11.62 s for the similarity search, and for the exact search they range from 0.20 to 4.54 s, which shows high variability due to real-world conditions.

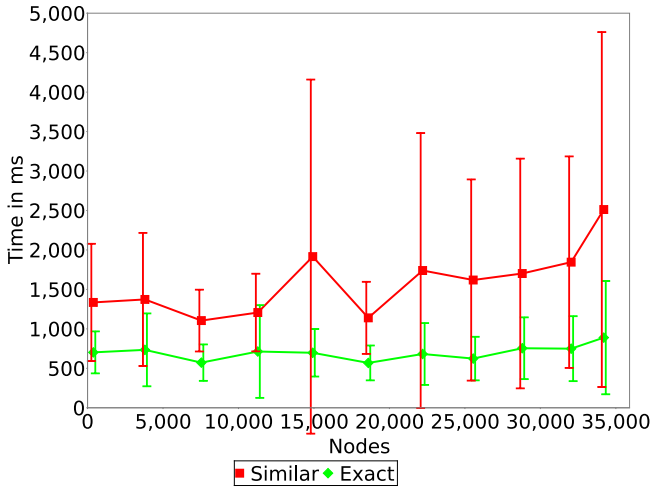

Fig. 10. Lookup time for similarity and exact search

The indexing time and standard deviation show a similar picture in Fig. 11. The values lie between 0.67 and 16.99 s for similarity indexing, and for exact indexing they range from 0.18 to 15.94 s. This shows that the storage operation is slower than searching, since keywords are stored with the redundancy factor *r*, which increases the number of messages that have to be exchanged

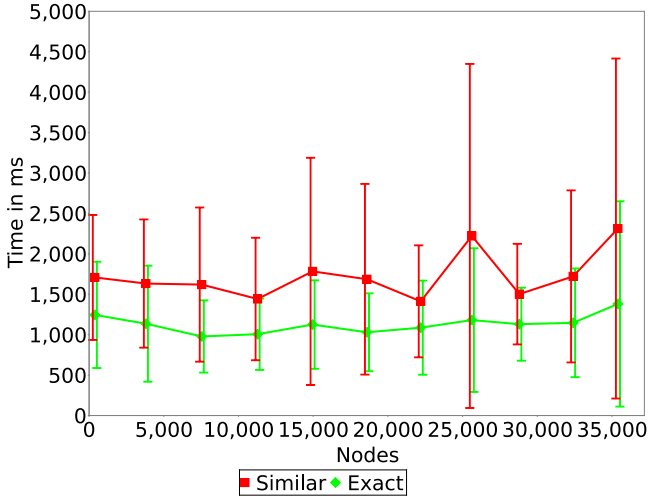

Fig. 11. Lookup time for similarity and exact indexing

During all experiments PlanetLab hosts had a load average of 9.9, which means that 9.9 processes requested 100% of the CPU time at any given time. Host availability decreased during the experiment. At the beginning 382 hosts participated, while towards the end there were only 342 hosts available. As a consequence, time measurements were higher, because unavailable hosts are detected by time-outs only. The difference in these time measurements between the similarity search and indexing as well as exact search and indexing is a result of the high average load, lack of availability, or high latency.

C. Storage Measurements

Storage usage is shown in Fig. 12. The number of keywords per node decreases as more nodes participate in the P2P network. This is expected, since the data is distributed on an increasing amount of nodes with a constant redundancy factor *r*.

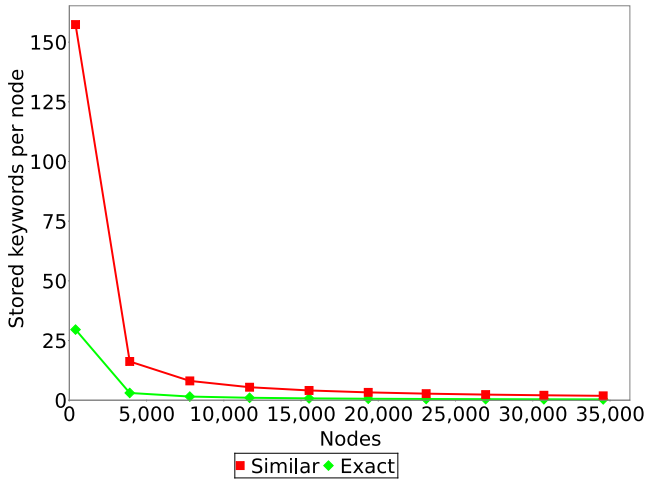

Fig. 12. Storage usage for similarity and exact indexing

VI. Discussion, Conclusion, and Futre Work

P2PFastSS offers similarity search based on the edit distance model, a type of search which is not efficiently supported by other solutions which are based on n-grams and cannot support queries on short keywords, as pointed out by [12]. P2PFastSS uses the Kademlia routing algorithm, however, it can be used on top of any P2P network, as it is independent of the DHT. This type of search provides a valuable alternative in service discovery, since full text-based discoveries for service descriptions will become more and more important in the future.

The experimental results show that a similarity search in a P2P network using the edit distance 1 is only 1.5 times slower than an exact search, while the message overhead is about seven times that of exact search. This difference is due to the benefits of distributed parallel computation. As P2PFastSS uses DHT operations in parallel, the measurements do not reflect isolated DHT operations. In effect, P2PFastSS performs a similarity search in less than 3 s with more than 34,000 nodes. The indexing process was carried out by a bootstrap node, to enable the collection of statistics, but in practice

it can be performed in a distributed manner, to improve performance.

All experiments described here used a set of Wikipedia abstracts, but similar results are expected with service descriptions in a service discovery scenario, if service descriptions and the abstracts have similar text length. Current results show that searches with edit distance 1 can be efficiently supported with P2PFastSS, since larger edit distances introduce a higher overhead. For example, storing a word with average word length of 7 and edit distance 2 would produce 3.625 times more messages than using edit distance 1. Further work will examine the performance of other FastSS variants and compression in data storage, as well as the use of other DHT overlay networks.

In summary, this paper demonstrates that P2PFastSS provides effective similarity search for text-based content in a timely and scalable manner, and warrants further investigation.

Future work will address the distribution of the load during the indexing process, which will improve the indexing time. Another important issue for future work is multiple keyword search and ranking. While a multiple keyword search can be implemented as multiple similarity searches, the ranking may be done based on service popularity or some other criteria, chosen dynamically. P2PFastSS will soon be deployed for service discovery with XML based service description, and the indexing framework will be integrated with other P2P networks.

ACKNOWLEDGEMENT

This work has been performed partially in the framework of the EU IST Project EC-GIN (FP6-2006-IST-045256) as well as of the EU IST NoE EMANICS (FP6-2004-IST-026854). E.H. is funded by an EU Marie Curie Fellowship. The authors acknowledge valuable comments from Pascal Kurtansky and from the anonymous reviewers.

REFERENCES

[1] K. Aberer; P-Grid: *A Self-Organizing Access Structure for P2P Information Systems*. Sixth International Conference on Cooperative Information Systems (CoopIS 2001), Lecture Notes in Computer Science, Volume 2172, pp 179-194, 2001.

[2] I. Aekaterinidis and P. Triantafillou; *Substring Matching in P2P Publish/Subscribe Data Management Networks*. IEEE 23rd International Conference on Data Engineering (ICDE), pp 1390-1394, April 2007.

[3] R. Ahmed and R. Boutaba; *Distributed Pattern Matching: A Key to Flexible and Efficient P2P Search*. IEEE Journal on Selected Areas in Communications, Volume 25, Issue 1, pp 73-83, January 2007.

[4] F. Banaei-Kashani and C. Shahabi; *SWAM: A Family of Access Methods for Similarity-Search in Peer-to-Peer Data Networks*. CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management, Washington, D.C., USA, pp 304-313, November 2004.

[5] M. Bawa, T. Condie and P. Ganesan; *LSH forest: self-tuning indexes for similarity search*. WWW '05: Proceedings of the 14th international conference on World Wide Web, Chiba, Japan, pp 651-660, May 2005.

[6] T. Bocek, E. Hunt and B. Stiller; *Fast Similarity Search in Large Dictionaries*. http://fastss.csg.uzh.ch/ifi-2007.02.pdf. Technical Report, University of Zurich, Number ifi-2007.02, April 2007.

[7] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana; *Web Services Description Language (WSDL) 1.1*. W3C, March 2001.

[8] CORBA/IIOP Specification. http://www.omg.org/technology/documents/formal/corba_iiop.htm, March 2004.

[9] R. Huebsch, B. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica and A. R. Yumerefendi; *The Architecture of PIER: An Internet-Scale Query Processor*. The Second Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA, January 2005.

[10] D. Hughes, G. Coulson and J. Walkerdine; *Free Riding on Gnutella Revisited: The Bell Tolls?* IEEE Distributed Systems Online, Number 6, Volume 6, pp 1, 2005.

[11] Java Remote Method Invocation. http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html, last visited: 09.09.2007

[12] M. Karnstedt, K. Sattler, M. Hauswirth and R. Schmidt; S*imilarity Queries on Structured Data in Structured Overlays*. ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops, April 2006.

[13] V. I. Levenshtein; *Binary Codes Capable of Correcting Deletions, Insertions and Reversals*. Soviet Physics Doklady, Number 8, Volume 10, pp 707-710, February 1966.

[14] W. Litwin, R. Mokadem, P. Rigaux and T. Schwarz; *Fast nGram-Based String Search Over Data Encoded Using Algebraic Signatures*. Very Large Data Bases (VLDB) 2007, Vienna, Austria, September 2007.

[15] P. Maymounkov and D. Mazieres; Kademlia: *A peer-to-peer information system based on the XOR metric*. In Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA, USA, March 2002.

[16] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein and S. Shenker; *Brief announcement: prefix hash tree*. 23rd annual ACM Symposium on Principles of Distributed Computing (PODC'04), St. John's, Newfoundland, Canada, pp 368-368, July 2004.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker; *A Scalable Content Addressable Network*. ACM SIGCOMM 2001, San Diego, California, August 2001.

[18] A. Rowstron and P. Druschel; *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pp 329-350, November 2001.

[19] I. Schmitt and S. Balko; *Filter ranking in high-dimensional space*. Data Knowledge Engineering, No. 3, Vol. 56, pp 245-286, 2006.

[20] G. Skobeltsyn, T. Luu, I. P. Zarko, M. Rajman and K. Aberer; *Web text retrieval with a P2P query-driven index*. SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, Amsterdam, The Netherlands, pp 679-686, 2007.

[21] I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan; Chord: *A Scalable Peer-To-Peer Lookup Service for Internet Applications*. ACM SIGCOMM 2001, pp 149-160, San Diego, California, August 2001.

[22] C. Tang, Z. Xu and S. Dwarkadas; *Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks*. ACM SIGCOMM 2003, Karlsruhe, Germany, pp 175-186, August 2003.

[23] UDDI Version 3.0.2: UDDI Spec Technical Committee Draft. http://uddi.org/pubs/uddi_v3.htm, October 2004.