Mobile P2P Fast Similarity Search

Thomas Bocek
Fabio Victora Hecht
David Hausheer
Department of Informatics IFI, CSG,
University of Zurich, Switzerland
Email: bocek—hecht—hausheer@ifi.uzh.ch

Ela Hunt
CIS, University of Strathclyde, UK
Email: ela.hunt@cis.strath.ac.uk

Burkhard Stiller
Department of Informatics IFI,
Communication Systems Group CSG,
University of Zurich
and TIK, ETH Zurich, Switzerland
Email: stiller@tik.ee.ethz.ch

***Abstract*—In informal data sharing environments, misspellings cause problems for data indexing and retrieval. This is even more pronounced in mobile environments, in which devices with limited input devices are used. In a mobile environment, similarity search algorithms for finding misspelled data need to account for limited CPU and bandwidth. This demo shows P2P fast similarity search (P2PFastSS) running on mobile phones and laptops that is tailored to uncertain data entry and uses available resources efficiently. In this demo, users publish and search for textual content containing misspellings without relying on query logging, as done by Google, and with a minimum distributed indexing infrastructure. Similarity search is supported by using the concept of deletion neighborhood to evaluate the edit distance metric of string similarity.**

I. Introduction

Few of the known distributed hash table (DHT)-based search methods support similarity search on keywords. Approximate keyword search is essential, as misspellings and spelling variants make the localization of required information a difficult problem. Without spelling correction, approximately 10% of all queries are not found, because of typos or misspellings [3]. Mobile devices usually have a limited keyboard for textual input, which makes misspellings more likely. Machine learning methods, as applied by Google are not always applicable, as they require a large corpus of queries.

In DHTs, the main operations are *put(key, value)* and *get(key)*. Those operations, in most cases, require $O(\log n)$ messages to be sent in a network with n nodes. Similarity search algorithms for structured P2P networks have been proposed by Ahmed et al. [1], introducing a routing algorithm based on Bloom filters and Wong et al. [6], introducing a routing algorithm in a keyword metric space. However, P2PFastSS [2] is the only algorithm that supports fast similarity searches that runs on existing DHTs without modifying the routing algorithm.

In a real world scenario, students on a large campus publish and index content and meta data, such as slides for the next lecture, from a specified directory on their mobile handsets. Content includes text files or media files with meta information. Other students that are on the same network search for content. Similarity search is necessary because of misspelled content and user queries.

This demo is a practical demonstration of the feasibility of P2PFastSS in a distributed mobile environment, and shows that P2PFastSS has great potential for real world applications. P2PFastSS provides a tradeoff between storage space and CPU time. While CPU time is minimized, storage space is required to store the deletion neighborhood. As reported in [2], the overhead of P2PFastSS in bandwidth for searching is 5.6 and for indexing 7 for a similar word of length 7, redundancy factor 5 and $k = 1$. Due to parallel execution, P2PFastSS is only 1.5 times slower than exact search. Thus, P2PFastSS is suitable for mobile environments because it reduces CPU time by using more storage space (storage space is cheap) and increases the bandwidth by a moderate factor, while keeping the overall search time low.

II. FastSS Algorithm

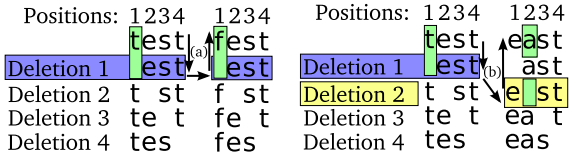
Edit or Levenshtein distance (ED) [4] can be applied to two sequences of symbols. It is defined as the minimum number of operations which transform one sequence into another. Operations are symbol insertion, replacement and deletion. Dynamic programming (DP) can be used to calculate the edit distance. However, with DP, indexing is not possible and a similarity search is carried out with linear complexity in the number of sequences.

FastSS uses an efficient variant of the neighborhood generation algorithm [5], adapted to use deletions only. This produces a smaller neighborhood which is then looked up in the index. For a given maximum ED=k, the results of all possible deletions in a given word are indexed. Deletions are applied recursively, and indices of deleted letters are ordered. For a word *test* with $k = 2$, the following relationships are indexed. Zero deletions produces *test* →*(test)*; one deletion produces *est* → *(test,1)*, *tst*→*(test,2)*, *tet* → *(test,3)*, and *tes* → *(test,4)*; and two deletions produce for each of the words with one deletion four variants, for instance for *est* → *(test,1)*, adding a deletion produces *st* → *(test,1,1)*, *et* → *(test,1,2)*, and *es* → *(test,1,3)*. Assuming natural language and a fixed edit distance ED=k, FastSS performs similarity searches with logarithmic complexity in the number of sequences [2].

Example 1: Two Deletions, Same Position Figure 1(a) shows that *d(test,1) = d(fest,1) = est*. *d(w,p)* stands for the transformation of word w by deletion of the letter at position p, *d(test,1)=est* and *d(test,2)=tst*. A replacement *t→f* at position 1 corresponds to two simultaneous deletions, and models one replacement, with ED(test,fest)=1.

Example 2: Two Deletions, Different Positions Figure 1(b) shows deletions at two different positions. *d(test,1) = d(east,2)*. In this case ED(*test,east*)=2.

Example 3: Single Deletion A single deletion for *east*, *d(east,2)* produces *est*. In this caseED(east,est)=1. As the deletion neighborhood is applied on the target and the source, the edit operation can be either a deletion or an insertion (*east* → *est*: deletion, *est* → *east*: insertion).



(a) Replacement *t*→*f* at position one of *test* to generate *fest*

(b) Deletion of *t* in *test* and deletion of *a* in *east* produce *est*

Fig. 1. Deletion neighborhood for *test* with *k=1*

III. Mobile P2P Fast Similarity Search and Demonstration Scenario

For P2PFastSS, the same concept of deletion neighborhood is applied and the neighbors from the target are stored in a DHT. Using P2PFastSS for the example *test* and *fest* in Figure 1(a) would result in the storage of *fest, est, fst, fet, and fes* using the put operation of a DHT.

The following example shows the indexing of the keyword *test* pointing to the document with DocID: 0x321. Keys in *get* and *put* operations are usually generated using a hash function $key = h(string)$. Node 0x1 first generates the neighbors of *test* with $k = 1$ (*test, est, tst, tet, tes*). All neighbors are indexed in the DHT. Figure III shows the indexing of *est*. First, node 0x1 looks for peers with an id close to the id of the neighbor where *est* should be placed, *h(est) = 0x123*. Node 0x4 replies in step 1 with the address of node 0x24, which is closer to 0x123. In step 2, node 0x24 replies with the addresses of nodes 0x124 and 0x122, which are closer to id 0x123. In step 3, these nodes will store the keyword and the document reference for *test*. The keyword is stored redundantly to provide robustness even in case of node failure.

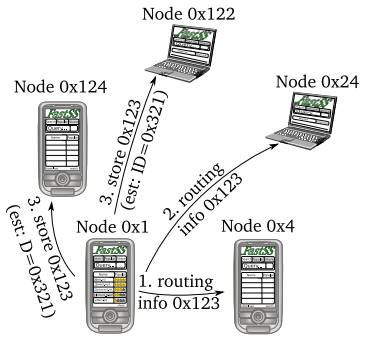


Fig. 2. Indexing of *est* with id=0x123

Figure III shows a query execution. Node 0x1 queries for the keyword *fest*. First, neighbors are generated (*fest, est, fst, fet, fes*). In steps 1 and 2, close nodes are queried for neighbor *est*, with id 0x123. In step 3, neighbor 0x124 which stores an index entry for neighbor 0x123 replies with the reference to document 0x321. This document contains the keyword *test*, and as $ed(test, fest) = 1$ the document or a preview of this document is shown to the user.

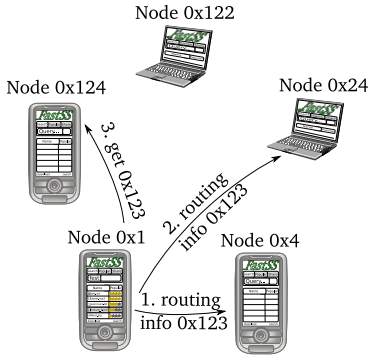


Fig. 3. Querying for *est* with id=0x123

The demonstrated prototype of mobile P2PFastSs implements the following three layers. The top layer is the user interface for document or content search. The undelying P2PFastSS layer offers two operations, *index* and *search* and carries out neighborhood generation, indexing and searching. The underlying DHT layer offers *get* and *put* operations. The DHT layer operates on top of Google's Android, which can store data persistently, play media files, and use WiFi.

In this demo, five users equipped with handsets share audio, video, and documents in a local network. As content, Wikipedia articles and multimedia files are used. Articles are written and indexed by users. Indexing video or audio requires meta information, a file name as a minimum. From the content, which is published on the local network, keywords are extracted and neighbors are generated. Each neighbor holds references to other devices, subject to a timeout. A user searches for content using keyword search. Movies, audio and pictures are searched for by filename or meta data, while text files are searched for by content. A keyword search may contain misspellings up to $k = 1$.

Acknowledgment This work has been performed partially in the framework of EC-GIN (FP6-2006-IST-045256) and EMANICS (FP6-2004-IST-026854).

References

[1] R. Ahmed and R. Boutaba. Distributed pattern matching: A key to flexible and efficient p2p search. *IEEE Journal on Selected Areas in Communications*, 25, Issue 1:73–83, January 2007.

[2] T. Bocek, E. Hunt, D. Hausheer, and B. Stiller. Fast similarity search in peer-to-peer networks. In *11th IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*, Salvador, Brazil, April 2008.

[3] H. Dalianis. Evaluating a spelling support in a search engine. In *NLDB*, pages 183–190, 2002.

[4] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, February 1966.

[5] E. Myers. A sublinear algorithm for approximate key word searching. *Algorithmica*, 12(4/5):345–374, 1994.

[6] B. Wong, A. Slivkins, and E. G. Sirer. Approximate Matching for Peer-to-Peer Overlays with Cubit. Technical Report http://hdl.handle.net/1813/10826, Cornell University, May 2008.