Search in Social Networks with Access Control

Truls A. Bjørklund
Norwegian University of Science and Technology
trulsamu@idi.ntnu.no

Michaela Götz
Cornell University
goetz@cs.cornell.edu

Johannes Gehrke
Cornell University
johannes@cs.cornell.edu

ABSTRACT

More and more important data is accumulated inside social networks. Limiting the flow of private information across a social network is very important, and most social networks provide sophisticated privacy settings to control this flow. Creating such extensive access control knobs makes the search for content a hard problem since each user sees a unique subset of all the data.

In this work, we take a first step at integrating access control based on a social network in a search system. We describe a set of solutions to the problem, including what indexes to construct and how to filter out inaccessible results. An experimental analysis illustrates the tradeoffs of the various strategies, and we point out a set of interesting future research directions in this area.

Categories and Subject Descriptors

H.3.4 [**Information Systems**]: Systems and Software—*Performance evaluation*

General Terms

Performance, Security, Experimentation

Keywords

Social Networks, Search, Access Control

1. INTRODUCTION

We consider the problem of keyword search in a social network where the network determines what data a user has access to. Search companies have already set their eyes on data in social networks: Google and Bing support search over Twitter posts. But due to privacy concerns, users are more and more limiting visibility of their data to only their social contacts. Facebook already allows users to search over the most recent posts of their friends, but the technical details are proprietary. Google also plans to include data shared by friends on social networks in search results.[1] Although there is obvious commercial interest in the problem of search in social networks with access control, we are not aware of previous academic work addressing it.

Enforcing access control in search engines is already supported in most desktop and enterprise search engines; a straight-forward solution is to create a separate index for each user, indexing only the documents accessible to that user. However, related work has shown that this approach does not scale to a large number of users because of the redundancy resulting from documents accessible by several users. Büttcher and Clarke suggest to build a global index for all documents and filter out results in a post-processing step [4], while Singh et al. propose to build one index per data collection with the same access permissions [11].

In this paper we describe the problem of keyword search in social networks with access control, and we make a first step towards a solution to the problem. We make the following contributions:

- We lay out a design space with two axes that exhibit beautiful symmetry: The first axis describes how to organize data into indexes, and the second axis describes how to organize access control information into author lists. (Section 3)
- In a thorough experimental evaluation with a real system and real and synthetic data, we show tradeoffs between important points in the design space. (Sections 4 and 5)
- We describe an exciting agenda for future research. (Section 7)

We discuss further related work in Section 6.

2. PROBLEM STATEMENT

Our goal is to support keyword search over content generated in a social network; we will refer to a piece of content both as a document and a post. Each document is *authored* by one user in the network. We view the social network as a directed graph (V, E). This captures both undirected graphs such as Facebook as well as directed graphs such as Twitter. Each node $u \in V$ corresponds to a user in the social network, and there is a directed edge $(u, v) \in E$ if user v is a friend of user u in the social network; we also say that u is one of v's *followers*. We denote by $F_u = \{v|(u, v) \in E\}$ the set of friends of user u and by $O_u = \{v|(v, u) \in E\}$ the set of followers of u.

[1] `http://www.networkworldme.com/v1/news.aspx?v=1&nid=3236&sec=netmanagement`

KEYS'10, June 6, 2010, Indianapolis, Indiana, USA.

A user has access to the documents authored by herself and to the documents authored by her friends. Thus, the result of a keyword query submitted by user u should include only relevant posts that (1) contain the search terms and that (2) are authored by a user in F_u. We call such keyword queries *queries with access control*, and since all queries considered in this paper are queries with access control, we will refer to them simply as *queries*. We rank the results of a query according to recency with more recent posts ranked higher than older posts; then we retrieve the top-k results. We can now (very informally) define the problem of keyword search in social networks with access control: Design a search system that enables fast queries with access control, efficient document updates, and that is also space-efficient.

We would like to emphasize that this problem definition makes several simplifying assumptions (for example, it assumes that the network structure is static); we will discuss extensions of this basic case when we discuss the research agenda in this space in Section 7.

3. DESIGN SPACE

Keyword search is usually enabled through an *inverted index*. Given a set of documents, an inverted index contains a *posting list* for each unique keyword in the documents. A posting list consists of *postings*; each posting describes an occurrence of the keyword in one document, including metadata used for ranking. Inverted indexes are the most commonly used data structure in today's search engines, and they have highly optimized implementations [14]. All solutions in this paper use the inverted index as a basic building block. We are now ready to describe the two axes of our design space.

3.1 The Index Axis

Our first axis is motivated by the observation that we can enforce access control by building not just a single inverted index, but a set of inverted indexes where each index contains all documents from a set of users. When processing a query from a user u, we do not need to involve indexes that do not contain documents authored by users in F_u. We can describe this more formally with the following two definitions.

Definition 1. (Group-Index) Given a set of users U, a *group-index* for U is an inverted index I that indexes all documents of users $u \in U$ and only documents of users $u \in U$; we say that $u \in U$ is a *member* of I.

For convenience, we will often identify an index I with its members U.

Definition 2. (Index Design) Let $G = (V, E)$ be a social network. An *index design* $\mathcal{I}$ for G is a set of group-indexes $\mathcal{I} = \{I_1, \ldots, I_k\}$ such that for each user $u \in V$ there exists a $j \in \{1, \ldots, k\}$ such that u is a member of I_j. We call k the *cardinality* of the index design, and we call the average number of group-indexes that a user is a member of the *redundancy* of the index design.

We can now characterize different index designs based on their cardinality and redundancy. Intuitively, a high cardinality has the advantage that for a user u there might be a set of group-indexes that closely "match" F_u; it has the disadvantage that we may have to combine the results from many indexes to answer a query. High redundancy implies both a large space consumption and poor update performance, but can again help to design group-indexes that closely "match" F_u for a user u with a small number of indexes.

There is a very large number of possible index designs. In this first paper, we only explore the space of extreme points to understand basic tradeoffs: We only consider cardinality 1 and $|V|$, and we only consider redundancy 1 and a, which is the average number of followers of a user. This gives rise to the following four index designs written as (cardinality, redundancy):

- (1,1): A single index stores all the documents; we call this design *global index*.
- (1,a): This solution is obviously suboptimal, and we will not consider it further.
- (|V|,1): One index per user u with the single member u; we call this design *user indexes*.
- (|V|,a): One index per user u with members F_u; we call this design *friends indexes*.

3.2 The Access Axis

Our second axis is motivated by the observation that we can enforce access control by storing explicitly which user has authored a document. We do this by creating *author lists* which contain pairs of *authorings*, where an authoring is a pair of (document identifier d, user identifier u), indicating that user u has authored document d. For a query by user u, an author list may allow us to filter the results from an inverted index: For each posting, we can check whether the document it occurs in was authored by a friend of u.

The design of author lists is our second axis, and it beautifully mirrors the design of the index axis. We can again define the notion of a *group-author list* as an author list containing all the authorings of a subset of users, and we can define an *access design* analogously as a set of group-author lists. Even the notions of cardinality and redundancy of a design carry over.

Analogously to index designs, there is a very large number of possible access designs. We again only explore the space of extreme points: We only consider cardinality 1 and $|V|$, and we only consider redundancy 1 and a, which is the average number of followers of a user. This gives rise to the following four access designs written as (cardinality, redundancy):

- (1,1): A single group-author list stores all the documents; we call this design *global-list*.
- (1,a): This solution is obviously suboptimal, and we will not consider it further.
- (|V|,1): One group-author list per user u with the single member u; we call this design *user-lists*.
- (|V|,a): One group-author list per user u with members F_u; we call this design *friends-lists*.

We would like to emphasize the nice symmetry between index design and access design; this gives a very clean characterization of the design space that we explore in our experimental evaluation.

3.3 Query Processing

Given an index design and an access design, we now explain how these two can be combined to answer a query submitted by a particular user u. We first select a set of

CPU	Intel Xeon (3.2 GHz)
Cache size	6 MB
Memory size	16 GB
Java version	1.6
OS	Red Hat Enterprise 5.3

Table 1: Experiment environment

group-indexes from the index design so that all friends of u are members of at least one group-index. If the set of group-indexes only contains postings from F_u and no other users, we can answer the query directly, and do not need to worry about the access design.

If the set of selected group-indexes have members $v \notin F_u$, we choose a set of group-author lists from the access design so that all friends F_u are members of at least one group-author list. If this selected set of group-author lists contains only members $v \in F_u$, then we only need to intersect the authorings in the group-author lists with the results of the query on the group-indexes on the document identifiers. If the group-author lists contain members $w \notin F_u$, then for each authoring, we need to check whether the author is a friend of u; this check can be performed in a lookup structure that stores all friendships in the graph.

4. IMPLEMENTATION

We have implemented a main-memory search engine in Java that supports the index and access designs outlined in Section 3. Postings resulting from a new document are accumulated in an updatable memory structure where postings are compressed using VByte [10], and become searchable immediately. The accumulated postings are routinely combined with the rest of the data in a hierarchy based on geometric partitioning [8]. More advanced techniques exist [7, 9], but they would not change the tradeoffs that we show. We use PForDelta [15] for compression, which has shown efficient decompression performance in recent studies [13]. For each index, we maintain a dictionary that allows us to look up term identifiers and retrieve the location and length of their posting lists.

Query processing is supported through a set of basic operators in addition to a *filter* operator used to filter the results of an intersection between a set of group-author lists and posting lists based on a lookup to test friendships. Together, these operators support the general query processing strategy outlined in Section 3.3. All operators work like Volcano-style iterators [6]. Their API has two methods: The GET NEXT method returns the next result from the operator while the SKIP method forwards to the next result with a given minimum value.

5. EXPERIMENTS

We have performed a thorough experimental evaluation with the system that we described in the previous section. Table 1 shows our hardware specifications.

5.1 Data

We explore the design space with workloads that are described in terms of networks, documents, and queries. We chose to first load all the documents before processing any queries, and measure the time it takes to load a document or to process a query separately; thus our results are easier to interpret than results for a mixed workload of queries and updates.

Networks. Our experiments use both synthetic networks and a real network from Twitter. The Twitter network was obtained by crawling roughly 417,000 users in February 2010. The synthetic networks of varying size and connectivity were generated using Barabasi's preferential attachment model [2].

Documents. All documents in our experiments were obtained from a crawl of Twitter. For the synthetic network we assigned Twitter documents to users by giving each user a posting frequency of a random Twitter user, and we assigned documents to users accordingly. This preserves the overall distribution of posting frequencies but not any correlation of frequencies and network structure. For the real network we retrieved the 200 most recent documents of the 417,000 users, and we then selected the 5 million most recent ones.

Queries. Since we do not have access to real search logs we generated synthetic query workloads as follows. For each search, we picked a random user u who searched for a random term occurring in the documents of u's friends. We process 100,000 queries returning the 100 top ranked results in all workloads.

Measurements. In all experiments, we report the average query and update times in addition to the space consumption. The space consumption is measured by collecting a dump of the complete memory heap when the index is loaded and garbage collection has been run. The reported results will therefore also include memory used in all configurations regardless of index or access design.

To explore both axes of the design space, we will first consider different index designs (Section 5.2), and then we focus on different access designs (Section 5.3).

5.2 Index Designs

In this section, we compare the performance of user indexes, friends indexes and the global index. The global index is combined with the global-list to enforce access control, while the other two strategies do not require an access design to enforce access control because all users can find a set of group-indexes that have exactly their friends as members. We also include a configuration with a global index without any access design as a baseline. Although this approach does not enforce access control, it enables us to illustrate the overhead of enforcing access control.

5.2.1 Scalability with Number of documents

The first experiment is based on a synthetic network with 1,000 users and 20 friends per user. The size of this network is so small to ensure that all index designs can handle it. We vary the number of documents between 25,000 and 150,000. The results are shown in Figure 1.

The search efficiency for friends indexes is clearly attractive and comparable to not enforcing access control because no unnecessary data is read or processed. User indexes, on the other hand, require combining results from 20 indexes during each search in this experiment, which makes them slower than friends indexes. The global index is even slower because many lookups to test friendship might be required before the results are found. When the number of documents increases, more and more documents will contain the search terms and more and more lookups to test whether the

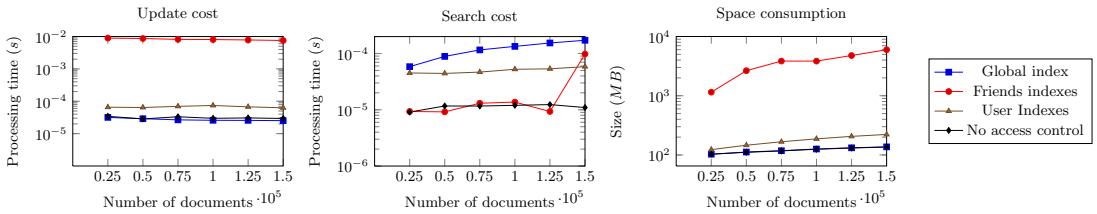

Figure 1: Time spent per update and search with different index designs (1,000 users, 20 friends per user)

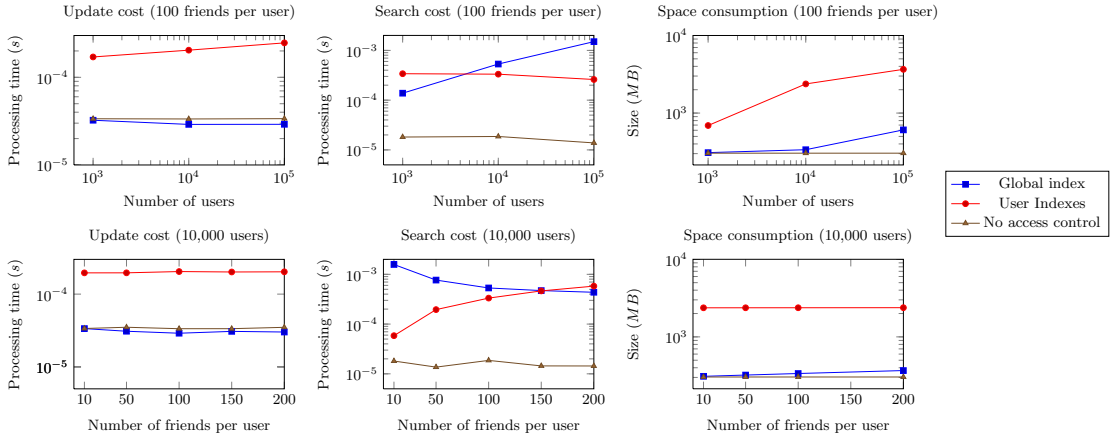

Figure 2: Results from varying network characteristics with different index designs (1,000,000 documents)

user who submitted the query is friends with the authors are then required.

While the friends indexes seem to be the most attractive design with respect to search performance, their huge size reduces performance with 150,000 documents due to swapping from memory to disk. The large size of friends indexes comes from the redundancy of indexing a document once for each of the author's followers. In addition to the increased number of postings resulting from the redundancy, the total size of the dictionaries also increases significantly because terms with a single occurrence will have one dictionary entry for each of the author's followers, while common terms will have one for each user. User indexes also suffer from larger dictionary sizes overall compared to a global index, because common terms will be represented in the index for many users, although the effect is small for the network size used in this experiment.

5.2.2 *Varying Network Characteristics*

The number of documents is not the only factor that determines performance, and we therefore conducted a set of experiments where we vary the network characteristics. After having established that friends indexes do not scale even to very small networks of 1000 nodes we focus on the remaining methods and move to larger input sizes. We vary the number of users in the network and the number of friends per user in two separate experiments. The number of documents is kept constant at 1,000,000. The results are shown in Figure 2.

The update cost for user indexes increases with increasing number of users, because the combined dictionary size is larger, which is also reflected in the index size. Although the update cost for the global index is constant regardless of network characteristics, the size of the lookup structures to test friendship increases with larger networks and with it the total space consumption.

When the fraction of documents that are accessible to the user who submitted the query increases, the search cost for the global index with global-list decreases because fewer look-ups are required before 100 results are found. The opposite effect is seen for user indexes; as the number of friends who have authored documents containing the search terms increases, their search cost also increases.

5.3 Access Designs

In this section we study the performance of the three access designs introduced previously. We test them in combination with the global index since neither user indexes nor friends indexes require an access design. Our conclusions apply more generally to any index design that relies on access designs to enforce access control. We also include a method without any access design for reference. The designs are tested with different workloads. We have excluded a workload with varying number of documents for access designs due to space restrictions, but experiment with varying network characteristics as we did for index designs, and also include a workload based on a real network.

5.3.1 *Synthetic Networks*

The experiments with synthetic networks are similar to the experiments with varying network characteristics for index designs except that the number of documents is kept constant at 2,500,000 and that we increase the number of users. The results are shown in Figure 3.

The update cost for all methods except friends-lists is comparable because the cost of updating the meta-data as-

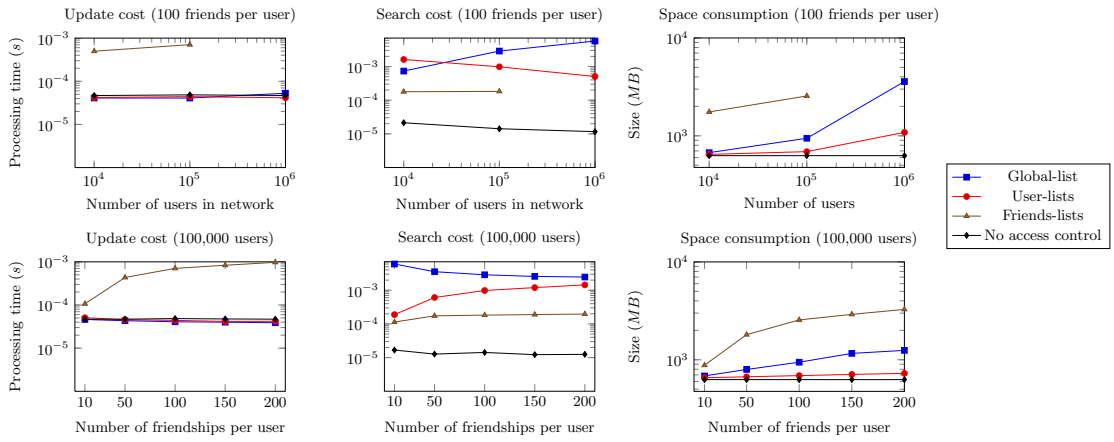

Figure 3: Results from varying network characteristics with different access designs (2,500,000 documents)

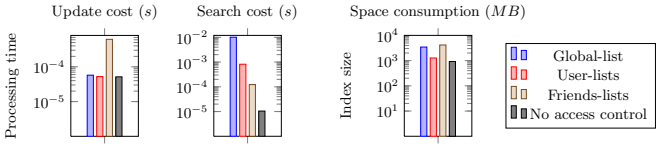

Figure 4: Results for real network

sociated with access control contributes only a small fraction of the overall cost (< 10%). For friends-lists, however, the index size grows linearly with the number of friendships in the network (creating an overhead for update costs of more than 2000% for 100,000 users with 100 friends per user) until we run out of space for 1,000,000 users. The space usage with the global-list is significantly higher than with user-lists for large networks because of the overhead associated with the lookup structure to test friendships. This structure is not required during query processing with the other access designs.

Friends-lists have an attractive search performance that does not depend on the size or the connectivity of the network. Filtering with friends-lists can still be 1500% slower than having no access control at all because instead of only scanning the 100 most recent postings in a posting list, we potentially have to skip forward in the lists many times. However, both user-lists and global-list are even slower and their performance depends on the network characteristics. Searches with user-lists become slower when the number of friends of the searching user increases because more group-author lists must be combined. The search cost with the global-list, on the other hand, decreases as the fraction of accessible documents increases because fewer postings must be processed to find the top 100 results.

5.3.2 *Real Networks*

The results from experiments with access designs on data from a real network are shown in Figure 4. The network used in the experiments contains roughly 417,000 users with 178 friends on average, and we generally observe the exact same tradeoffs as in the synthetic networks with similar sizes. Because the number of documents is scaled up in this experiment, each update is slightly slower due to more expensive merges in the hierarchy of indexes as the index size grows. In this particular network, the average number of followers for the author of each document is slightly lower than in our synthetic networks with the same size, which makes updates for friends-lists slightly faster here relative to the other approaches. The search cost is also affected by the increased number of documents, because each query will on average have more results.

5.4 Discussion

Our experiments show that our designs represent different tradeoffs between index size, update and search performance. The designs along the two axes reveal similar tradeoffs which is due to the symmetry between the axes. However, the differences between the solutions for different index designs are generally much larger than between the solutions for different access design.

Based on our experiments, we believe that user indexes, or a global index with either friends-lists or user-lists as the access design is the best of the basic solutions in real-world scenarios. We do not recommend to use friends indexes because of their scalability problems. The choice between user indexes, or a global index with either user-lists or friends-lists as the access design should be made dependent on the expected workload of updates and searches, the network structure and possible space constraints.

6. RELATED WORK

For related work on access control models for structured data we refer the reader to the excellent survey by Bertino et al. [3].

The problem of enforcing access control occurs in both desktop and enterprise search systems, and a straight-forward solution is to create a separate index for each user, indexing only the documents accessible to that user. In a social network we introduced this design as friends indexes. Related work has shown that this approach does not scale to a large number of users because of the redundancy resulting from shared documents, a fact that is confirmed by our experiments. Several alternatives have been suggested [4, 11]. Singh et al. propose to group files based on their access permissions, so that all files in a group are accessible to the same users [11]. Each group is indexed and each user forwards a

search to a specified set of indexes. When a social network determines the access permissions, the approach would typically yield an index design close to user indexes, because most users have a unique set of followers. Singh et al. also describe how the set of indexes can be reduced when introducing redundancy, and they also mention that it is possible to filter out results, just like we do with different access designs. However, they do not consider these solutions relevant in their usage scenarios and do not explore them. Büttcher and Clarke suggest to enforce access control by filtering results from a global index while making sure that ranking statistics are based solely on information accessible to the user conducting the search [4]. The exact filtering strategy is not described in detail. Bailey et al. describe how filtering can be integrated into an overall enterprise architecture [1]. Our work can be seen as an extension that explores different access designs which can be interpreted as different filtering strategies, and our experiments indicate that the choice of access design has a significant impact on performance.

Zerr et al. consider security attacks on enterprise search architectures where a certain fraction of the servers is compromised, and propose a system that limits the amount of information leakage in such scenarios [12], an orthogonal problem to what we consider in this paper.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have taken the first steps towards addressing the problem of search when a social network determines the access permissions. To do so, we have considered keyword search over documents in the social network, and we outlined a symmetric design space consisting of index designs and access designs. Our experiments with several basic solutions indicate that user indexes or global indexes with either user-lists or friends-lists as access designs are the most promising solutions in real-world scenarios.

We believe that the problem we have addressed in this paper only scratches the surface of an important research area for future work. The design space of both axes is wide open and requires further exploration beyond the basic strategies we have evaluated in our experiments. Identifying the best strategy for a particular workload is an important direction of future work.

Challenges on the systems-side include the scalability to large networks and adaptability to their dynamic structural changes. A search system for a large network will probably have to be distributed in order to scale, and partitioning users and their data across machines is an open problem.

Extensions to support more advanced ranking functions are important, possibly including functions that take the actual social network structure into account (see [5] for an example). With more sophisticated ranking functions, it is important to avoid that the ranking reveals protected information to users [4, 11].

The overall problem of search in social network goes well beyond keyword search. Documents are not the only entities that could be searchable in a social network, and extensions to allow queries over structured data in a social network is another avenue of future research.

Acknowledgments. This material is based upon work supported by the New York State Foundation for Science, Technology, and Innovation under Agreement C050061, by the National Science Foundation under Grants 0121175 and 0534404, and by the iAd Project funded by the Research Council of Norway. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

8. REFERENCES

[1] Peter Bailey, David Hawking, and Brett Matson. Secure search in enterprise webs: tradeoffs in efficient implementation for document level security. In *Proc. CIKM*, 2006.

[2] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[3] Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. Database security: research and practice. *Inf. Syst.*, 20(7), 1995.

[4] Stefan Büttcher and Charles L. A. Clarke. A security model for full-text file system search in multi-user environments. In *Proc. FAST*, 2005.

[5] David Carmel, Naama Zwerdling, Ido Guy, Shila Ofek-Koifman, Nadav Har'el, Inbal Ronen, Erel Uziel, Sivan Yogev, and Sergey Chernov. Personalized social search based on the user's social network. In *Proc. CIKM*, 2009.

[6] G. Graefe. Volcano – an extensible and parallel query evaluation system. *IEEE Trans. on Knowl. and Data Eng.*, 1994.

[7] Sairam Gurajada and Sreenivasa Kumar P. On-line index maintenance using horizontal partitioning. In *Proc. CIKM*, 2009.

[8] Nicholas Lester, Alistair Moffat, and Justin Zobel. Fast on-line index construction by geometric partitioning. In *Proc. CIKM*, 2005.

[9] Giorgos Margaritis and Stergios V. Anastasiadis. Low-cost management of inverted files for online full-text search. In *Proc. CIKM*, 2009.

[10] Falk Scholer, Hugh E. Williams, John Yiannis, and Justin Zobel. Compression of inverted indexes for fast query evaluation. In *Proc. SIGIR*, 2002.

[11] Aameek Singh, Mudhakar Srivatsa, and Ling Liu. Efficient and secure search of enterprise file systems. In *Proc. ICWS*, 2007.

[12] Sergej Zerr, Elena Demidova, Daniel Olmedilla, Wolfgang Nejdl, Marianne Winslett, and Soumyadeb Mitra. Zerber: r-confidential indexing for distributed documents. In *Proc. EDBT*, 2008.

[13] Jiangong Zhang, Xiaohui Long, and Torsten Suel. Performance of compressed inverted list caching in search engines. In *Proc. WWW*, 2008.

[14] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 2006.

[15] Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. Super-scalar ram-cpu cache compression. In *Proc. ICDE*, 2006.