ACCESS RIGHTS IN ENTERPRISE FULL-TEXT SEARCH
Searching Large Intranets Effectively using Virtual Terms

Jan Kasprzak, Michal Brandejs, Matěj Čuhel and Tomaš Obšivač
Faculty of Informatics, Masaryk University, Brno, Czech Republic

Keywords: Full-text search, Intranet, Security, Access rights, Document permissions, Document-level security.

Abstract: One of the toughest problems to solve when deploying an enterprise-wide full-text search system is to handle the access rights of the documents and intranet web pages correctly and effectively. Post-processing the results of general-purpose full-text search engine (filtering out the documents inaccessible to the user who sent the query) can be an expensive operation, especially in large collections of documents. We discuss various approaches to this problem and propose a novel method which employs virtual tokens for encoding the access rights directly into the search index. We then evaluate this approach in an intranet system with several millions of documents and a complex set of access rights and access rules.

1 INTRODUCTION

Many enterprises have put effort in developing web-based authenticated systems, covering many aspects of internal processes, called *intranet systems*. Some intranets provide not only access to various structured pieces of information (such as list of members of some work group, or a list of occupied time slots of a meeting room), but also unstructured *text documents* (such as study materials for the university courses, whitepapers, internal product documentation, etc.).

It is often difficult for users to navigate inside a large web-based intranet system. Full-text search can provide an alternative (or even primary) means of navigation (Zhu et al., 2007). The task of searching the intranet system effectively has many problems which are completely different to the problems of searching the publicly-available world-wide-web (Hawking, 2004).

An important problem is to be able to follow the access rights of the web pages and documents effectively: the intranet systems often contain documents and pages, which are not permitted to be read by every authenticated user, so it is not possible to allow them to be discovered using a full-text search system. In this paper we propose a solution to the access rights problem in the enterprise full-text search, and we evaluate this approach on Masaryk University Information System (IS MU, 2010), a large intranet system for study administration.

The rest of this paper is organized as follows: in Section 2 we discuss related work, in Section 3 we present an overview of general-purpose full-text search engines, especially with respect to our proposed method. In Section 4 we discuss access rights management. The main contribution of this paper is in Section 5, where we propose a novel method of incorporating security features to the full-text search system. We further evaluate this method on a real world intranet system with tens of thousands users and several milion documents and pages in Section 6. Finally, the Section 7 contains conclusions.

2 RELATED WORK

Several authors have attempted to deal with issues concerning full-text search in intranet systems, where the access rights to documents have to be taken into account.

In (Bailey et al., 2006), the two architectures of enterprise systems are presented. In the first one the search engine is not involved in security policies or access rights: users communicate with a repository, which forwards the query to the search engine, and then filters out the documents inaccessible to the user. In the second approach the search engine handles the access control by querying an external service such as LDAP. The cost of security checks in the second approach grows linearly with the number of results to check, leading to query times of 100 seconds when

Kasprzak J., Brandejs M., Čuhel M. and Obšivač T. (2010).
ACCESS RIGHTS IN ENTERPRISE FULL-TEXT SEARCH - Searching Large Intranets Effectively using Virtual Terms.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 32-39
DOI: 10.5220/0002896900320039

the number of candidate results reaches 1 million documents, even when caching the access control data.

In (Hurley, 2009), the author realizes that filtering out the results in the post-processing phase may have negative impact on the search performance. A different approach is presented, where the users' credentials are calculated for every user when the document is being indexed. A per-user view of the index is created, and subsequently used for evaluation of user's queries. To be able to maintain consistent copy of access control lists, quick check to the directory service is performed before each query. Two implementations are presented: *PrivaSearch1*, where each user has specific view of the corpus, and *PrivaSearch2*, where the membership in groups is also reflected. As a result of this method each user is provided with a separate index. Three types of document distributions are discussed: disjoint, overlapping, and hierarchical. The implementation is tested on data sets of 2,000 to 20,000 documents. For larger data sets, the author suggests to distribute the index. We consider this approach (per-user indexes) completely feasible only for a limited range of systems where users have access to disjoint sets of documents (e.g. e-mail messages).

Google Search Appliance (GSA, 2009) is as a widely known commercial black box solution for intranet or website full-text search. It performs on-line access control checks for each document from the result set. Standardized SAML (SAML, 2005) identity provider (or specialized one for batch processing) has to be present or implemented on the customer side.

The post-processing approaches have limitations on large scale systems. For example, when there is a thousand of higher-ranked documents matching the query, but inaccessible to a given user, the system will need to check the access rights to all of those documents before finally discovering a lower-ranked document accessible to that user.

3 FULL TEXT SEARCH ENGINES

The full-text search systems are often constructed as a software, which maintains the *index* of the documents to be searched, and executes queries against this index. We refer to (Zobel and Moffat, 2006) for possible approaches for creating and using the index.

3.1 Index

For the method we propose, it is sufficient to view the full-text search engine as a maintainer and user of an *inverted index*, consisting of *lexicon* of all words from the set of documents, and mapping of each of those words to the *list of document IDs*, listing the documents which contain that word.

These lists are often sorted by the document ID, allowing to store them differentially, and compressed using e.g. Elias delta encoding (Elias, 1975). They also might be accompanied by *weights* of the given word in a particular document.

For evaluating multi-word proximity and phrase queries, it may be useful to allow fast forward searching of these lists, which can be done e.g. by accompanying larger lists with skip-lists for semi-direct access.

The index also contains other data structures, which are not relevant to this paper (forward index, lists of word positions, etc.).

3.2 Query Format

The query itself is usually entered by user as a sequence of words, meaning "find the documents which contain *all* of those words, preferably near to each other" (i.e. the implicit AND/NEAR operator). Many search engines allow also the NOT operator, and some of them support also the OR operator. The support for exact phrase searches is common as well.

The OR operator—even when not available directly to the end user—is often used internally for handling lemmatization, inflections, diacritics, acronyms, and so on. For example, the query "*Citroën cars*" can be internally transformed as follows:

(citroën OR citroen) AND (cars OR car)

For our purposes, it will be sufficient when the search engine can efficiently handle the queries in the above format—i.e. the logical conjunction (AND) of the logical disjunctions (OR). More precisely formulated, the expected query format is the following:

$$\begin{array}{ll} & (token_{1,1} \text{ OR } token_{1,2} \text{ OR } token_{1,3} \text{ OR } \cdots) \\ \text{AND} & (token_{2,1} \text{ OR } token_{2,2} \text{ OR } token_{2,3} \text{ OR } \cdots) \\ \text{AND} & (token_{3,1} \text{ OR } token_{3,2} \text{ OR } token_{3,3} \text{ OR } \cdots) \\ & \vdots \end{array} \quad (1)$$

4 ACCESS RIGHTS

The availability of a document (or, more generally, *object*) to some user (i.e. *subject*) is often described by a matrix with per-subject rows, per-object columns, and cells containing the actual permissions settings (a subset of "can read", "can write", "can delete", etc.). For the full-text search, only the read permission is significant. Note, however, that some systems

(e.g. LDAP) can have a separate “can read” and “can search” levels of access.

Storing the matrix as a whole is often impractical, so the systems (be it web-based intranets or operating systems) often store the parts of the matrix either as rows, often called *capabilities* (e.g.: “user *root* can read every file”, or “the teacher can read all of the study materials of his own courses”), or as columns, called *access control lists* (ACLs) in the Chapter 5 of (Anderson, 2008)—e.g. “this document can be read by members of group *staff*”, or “this document can be read by all present and past students of the course ‘UNIX’”.

Some systems store the access rights information explicitly (e.g. filesystems), while other systems have some of the implicit rules hardcoded inside the system (this is the case of many enterprise systems).

To reduce the size of the access rights matrix (stored either as capabilities or ACLs), subjects are often organized into *groups* (e.g. group *staff*, or “students of the course ‘UNIX’” from the above examples).

For the approach we propose in Section 5 it is sufficient to be able to evaluate the following two lists:

- the list of groups a given user belongs to,
- and the list of groups which can read a given document.

As a last resort, if no smarter method of grouping is available, it is possible to put every user in his own separate group.

5 PROPOSED METHOD

The high-level view of the proposed system architecture can be seen in Figure 1. Some components are present in general intranet systems as well:

SQL database stores structured data of the intranet system. It is often encapsulated by a middleware layer.

Document Storage. Large static text and multimedia files are usually stored outside the database.

HTTP Front End. This is the system which the user’s browser communicates with. It handles authentication, parsing the HTTP requests, and computing the resulting pages.

The following components are added for the purpose of full-text search:

Crawler/indexer downloads the static documents from the document storage, and either downloads the dynamically generated pages from the HTTP server, or constructs them from data in the database the same way as the HTTP server does. The documents are then parsed and added to the index.

Search Index. Contains the data structures described in Section 3.1.

Search Server. Receives the pre-parsed queries from the HTTP server, executes them using the pre-computed index, and sends the results back to the HTTP server, which then formats them to the user.

Note that the search server does not communicate with the SQL database, so the pre-parsed query can be executed using the search index as the only source of information.

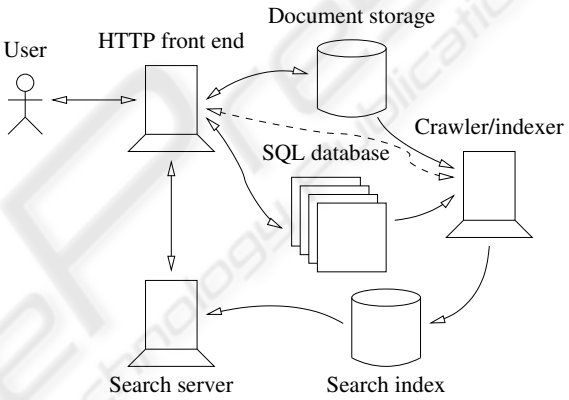

Figure 1: Architecture of an intranet search system.

5.1 Virtual Tokens

We propose to enhance the search engine with support for what we call *virtual tokens*: virtual token is a special word, added to the inverted index, as if it was a part of the indexed document itself. The difference between words an virtual tokens is the following:

- The virtual token has no position within the document. It is thus excluded from either the proximity or exact phrase searches.
- It has no weight, so the search results cannot be sorted using the weight of a virtual token. Another (non-virtual) token has to be present in a query.
- The virtual token in one document cannot be an ordinary word in another document. This is easily satisfied by adding a character which can never be a part of a word. We use the colon (:) for this purpose.

The above requirements allow us to handle the virtual tokens inside the inverted index differently: the list of document IDs for a virtual token does not

need to include anything beyond the (differentially-encoded and delta-compressed) document IDs themselves. No pointer to the list of positions and no weight of a word in a given document is needed.

5.2 Access Rights Encoding

The access rights (be it ACLs or capabilities) are then named and encoded as groups of users (e.g. "teachers of the 'UNIX' course" or "students of the 'Informatics' study programme").

The permissions of a given document are represented as a set of groups which have the read access to the given document. This set can be then encoded into the inverted index as virtual tokens—derived for example from the name or the number of a given group (e.g. "*p:teachers_UNIX*", or "*p:programme_Informatics*"; the "*p:*" prefix we reserve for virtual tokens describing the access rights). For each group, members of which can read a given document, we add a virtual token with the group identification for this document to the inverted index.

When the document should be accessible without authentication (i.e. when it is world-readable), we add a reserved virtual token "*p:noauth*" to it. When the document should be available to any authenticated user, we add another virtual token "*p:auth*" to it.

5.3 Query Execution

The user's query is parsed by the HTTP server to the format (1). To limit the search results to the documents which the user has permission to read, the HTTP server computes the list of all groups the user belongs to (for non-authenticated user, the list consists of the single group "*noauth*"). It then modifies the query by adding another OR-list the following way:

$$\begin{aligned} &\textit{The\_preparsed\_query} \\ \text{AND} \quad &(\text{p:}\textit{group}_1 \text{ OR p:}\textit{group}_2 \text{ OR } \cdots) \end{aligned} \qquad (2)$$

With this part added, the search server returns only those documents matching the user's query, which are also readable by this user.

5.4 Expected Performance

The proposed system has three places, in which the overall performance is affected by adding the access rights:

Indexing the Document. This part of the system runs in a batch mode, so the interactive latency of the system is not affected. As for overall performance, we expect the evaluation of the document's access rights to have negligible cost relative to e.g. tokenizing the text or converting the document from formats like PDF or DOC.

Evaluating the Per-user Group List. This can be either very fast or relatively expensive operation, depending on the format of the source data. For example, fetching the list of groups from the UNIX `/etc/group` table or from LDAP can be fast, while computing the list from the implicit intranet data can be expensive (for example, for study materials, a separate group for each course and possibly semester has to be created, and the list of courses the user has ever enrolled in should be evaluated). Should this operation be expensive, we can cache the resulting list of groups for a given user for some time (a day, or possibly an hour). We can even precompute the list on background after displaying the full-text search form in order to improve the latency.

Executing the Qquery. The impact in this part can potentially be very high: the search server needs to process many and/or long additional lists of document IDs for many virtual tokens. However, with some optimizations, the search can theoretically be even faster than without the access rights. For example, when the search server processes the OR-list with the lowest number of matching documents first, executing the query for a very frequent word can be faster for users which can read only a small number of documents. On the other hand, when the query is executed on behalf of user who can access large portion of the available document base, we can optimize the query execution by adding skip-lists to the large lists of document IDs. The impact is further lowered by simplified lists for virtual tokens, as described in Section 5.1.

The main advantage of this system is that it does not require the access rights of the individual documents to be evaluated in the query results post-processing. This helps especially when the search results contain lots of documents which are highly relevant, but inaccessible to a given user. With our approach, we are able to filter out the inaccessible documents during the query execution phase the same way multi-word queries work.

5.5 Drawbacks and Limitations

The system we have just described—like many other computing systems—contains a tradeoff between the efficiency and accuracy. We will try to identify and

describe the possible drawbacks of using such a system:

The Permissions are "Cached" in the Index. This means that the system can sometimes allow the document to be found even after the access for a given user has been revoked. From our point of view the problem is not so critical: for urgent cases (such as when the user can no longer be trusted) the common approach is to revoke the access to the intranet system as a whole, so the full-text search system is also no longer accessible.

For revoking the permissions for a given *group*, the situation is the same as when removing a part of the document in a new version of the document: we do not expect the immediate reindexing the document when modifying its contents, so we should not expect the immediate revoking of the rights as well. In our experience, it is feasible to invoke the reindexing of the modified documents frequently (e.g. every minute), so the delay of propagation of modified access rights can be kept relatively low. Also note that in this case we are trying to deny a read access to something which could have been readable for a long time now.

Negative Access Rights. Some systems of access rights can use the group membership also to *lower* the permissions for a given user. For example, the classical UNIX access rights allow the file to have permissions set to `rw----r--`, which means that the file is readable and writable by its owner, and readable by everybody *except* the members of the group the file belongs to. This can be mapped to our approach only by a new virtual token/group "*all users except members of the group XY*".

Tree-like Rights are not Straightforward. It is also possible for documents to be organized in a directory tree, in which not only the permissions of the document itself are evaluated, but also the permissions of the directories in the path from the document to the root of the tree. The problem arises when the subdirectory or a file has permissions, which are not a subset of the permissions of the upper directory. For example, the directory can be readable by a group "*students of the programme 'Informatics'*", and the document in it can be readable by a group "*students who have enrolled in the course 'UNIX'*". Since the first group is not necessarily a superset of the second one, we should effectively provide the access only to the users which are members of both of the above groups. This also can be solved by creating another virtual token/group "*students of the programme 'Informatics', who also have enrolled in the course 'UNIX'*". This approach requires additional work when evaluating the list of groups for a given user, but it can be implemented.

Despite the above drawbacks, we believe that our approach is usable for many intranet systems, as it can be faster than on-line verification of the access rights.

5.6 Other Uses of Virtual Tokens

Virtual tokens as described in Section 5.1 can be used for other purposes beyond access rights: the document date can be encoded into virtual tokens, and then user can specify the time range. For example, the following query can select the time span of 13 months starting with January 2009:

The_preparsed_query AND (d:2009 OR d:201001)

It can also be used for improving the search engine relevance: we can add weight to the documents containing a given virtual token instead of filtering them out when the required virtual token is not present. It is then possible to give preference to the study materials of courses attended by student, even when other courses may have their study materials publicly accessible.

Advanced search forms can specify additional search criteria like increased weights for documents in a given subtree. Another possible use of virtual tokens is in search log evaluation (e.g. finding terms most searched by students of the 'UNIX' course).

6 CASE STUDY

We have tested our approach on a real-world intranet system with relatively complicated system of access rights.

6.1 About IS MU

The Masaryk University Information System (IS MU, 2010) is a study administration and e-learning system. It has been developed since 1999. More than 130,000 users—present and past students and employees of the university—can access the system. As of January 2010, the system is accessed by more than 30,000 unique users daily, and handles more than 2,000,000 authenticated HTTPS requests per day. The document storage contains over 20,000,000 objects (documents in various formats, images, multimedia data, theses, e-mails, etc.).

6.2 Access Rights System

The system of access rights in IS MU consists of the *explicit* access rights, which can be set by the owner of the document (e.g. "*this document can be read by the students of the course "UNIX"*"), and *implicit* access rights, which are mostly hardcoded in the applications or PL/SQL procedures. This includes things like "*the study materials of a given course are irrevocably readable by all teachers of that course*", or "*the permissions of the discussion forum posts cannot be set at all, the permissions are derived from the top-level node (representing the forum itself)*", etc. There are several types of explicit rules (based on the course, study programme, workgroup membership of an employee, faculty, alumni status, type of study, etc.).

6.3 Testing Platform

We have implemented the access rights handling method as described in Section 5 into the IS MU full-text search system. The searcher and indexer run on a single server SGI Altix XE (a rebranded SuperMicro) with dual Xeon E5472 CPUs at 3.0 GHz (8 cores total), 64 GB RAM. The system runs 64-bit version of Fedora Linux.

The tokenization of documents (including evaluating the access rights) has been implemented in Perl, creating the partial inverted index, merging indexes, and searching were implemented in plain C with Judy arrays (Judy, 2010) for storing associative data such as lexicon. The HTTP front-end applications are written in Perl: pre-parsing the queries including evaluation and caching the per-user list of groups runs there.

The index is segmented into 8 parts of similar size (per-document split), and during the query execution each CPU core works on top of its own part of index. The partial results are then merged together.

6.4 Users

We have decided to test the system on a subset of users, who have used the system during October 2009 (October is one of the busiest months of a year for the system, because of the start of semester). There are 51,022 such users, which form 768,442 different groups. Each user is in at least two groups ("all authenticated users", and a per-user group). The median is 178 groups per user, mean is 210, and the maximum is 9,942 groups per user. The histogram of the number of users with a particular number of groups (without the long tail of users which are members of more than 800 groups) is in Figure 2.

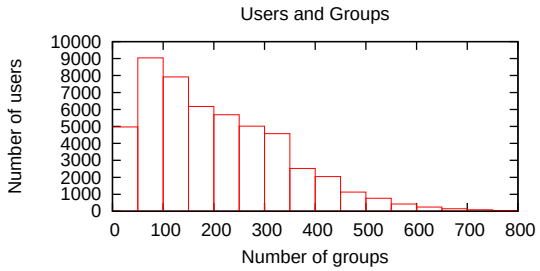

Figure 2: Number of users with a given number of groups.

The median of the time for evaluating the per-user list of groups was 0.028 s, mean 0.048 s, and maximum was 1.415 s (we have took the minimum from five runs). The tests were run on a production database without exclusive access to the computing resources.

6.5 Documents

As a testing set we have used 1,370,200 documents from various *agendas*, both static (e.g. theses and study materials), and dynamically generated (e.g. personal home pages). The documents are accessible by 60,493 different groups (this number is much smaller than the total number of groups for all users, because e.g. some courses contain no study materials at all). The largest number of documents a single group can access is 1,129,995. Another important groups are "non-authenticated users" which is the third with 398,391 documents, and "all authenticated users" pseudo-group (14th, 107,520 documents).

Further optimizations of the group memberships in IS MU are possible, but are outside of the scope of this article.

The performance of the indexer depends highly on the document size. For example, for theses, the system can tokenize 9 documents per second on average, while for the rest of the testing set (including the study materials) the speed is around 80 documents per second. The tokenizing and generating the inverted index of the whole document set takes slightly more than 6 hours. This time does not include converting various foreign formats (DOC, PDF) to plain text.

The resulting index occupies about 18.3 GB. The data relevant for our test (lexicons and document lists for all words) occupy 2.7 GB. The biggest part of the index is forward index for generating document snippets (7.8 GB) and word position lists for phrase searches (7.7 GB).

The overhead of adding the virtual tokens is in the lexicon, where it adds 60,493 more words to the 16,868,473 words which were already present, and in the lists of document IDs for those words, where it costs additional 8,449,607 list entries to the already

existing 932,424,699 entries. We can say the index size overhead is less than 1 %.

6.6 Search Performance

In this section we evaluate the performance of the search server as described in Figure 1. One of the most computing-intensive parts of the query execution is decoding the lists of document IDs. We have measured that with our implementation, a single core of the search server can decode about 55 millions of differentially-encoded delta-compressed numbers (with up to 64-bit size) per second.

We have tried to measure the pre-parsed query execution times for various single-word queries with and without applying the access rights. We do not include multi-word and phrase queries here for brevity.

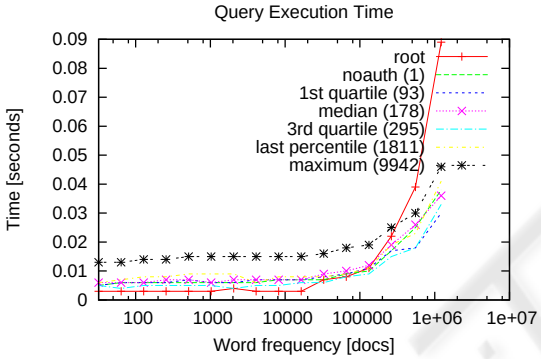

Figure 3: Query execution times for various users and terms of a given frequency.

Another difference from the production use is that the search server did not generate the text snippets of the resulting documents, as this part is not relevant to the measurement of the access rights overhead.

We have chosen a set of words such that the words are present in approximately power-of-two number of documents, and are English words, because of the language of this paper. The most frequent word is '*a*', which is present in 1,221,642 documents. The other words in descending frequency are *7, on, role, both, party, speech, warning, movies, broadest, peptides, initiators, prefect, realigning, prescript*, and finally *noncyclic*, which is present in 32 documents.

We have measured the query execution time for the above words for a sample set of users. Firstly—as a baseline—we have the search without applying the access rights at all (superuser access), then a non-authenticated search (i.e. limited to the group "*non-authenticated requests*"), and then as users with 93, 178, 295, 1811, and 9942 groups (first quartile, median, third quartile, last percentile, and maximum; from the users with similar number of groups the ones

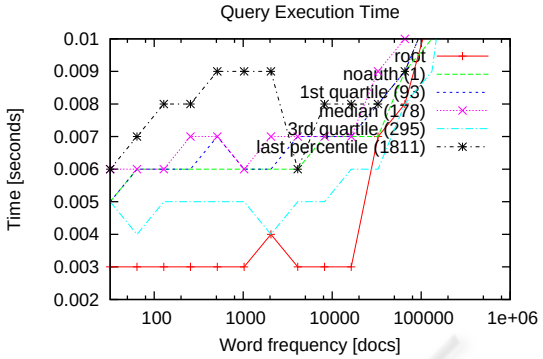

Figure 4: Detailed view on the query execution times.

with maximum number of accesses during October 2009 have been chosen, in order to simulate the real-world performance for more active users). We took the lowest time from 10 runs for each query. Note that generating the per-user group lists, as described and measured in Section 6.4, is not included in these times.

The Figure 3 shows the execution times of queries. Figure 4 shows the same graph with limited range of the vertical axis to highlight the details. Finally the Figure 5 shows the overhead relative to the search without applying the access rights.

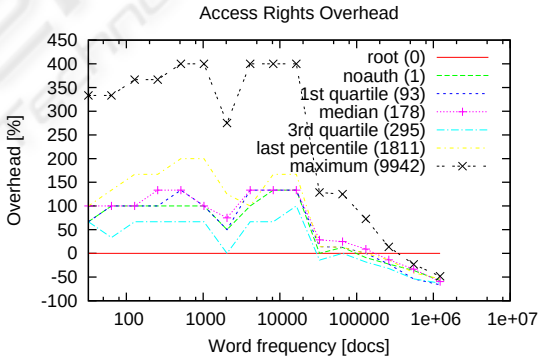

Figure 5: The overhead of following the access rights.

6.7 Discussion of the Results

The cost of generating per-user group lists in IS MU is non-trivial and introduces a significant latency. We have therefore added caching of the lists for a day to the production system. However, the system is not typical because of large number of groups and complicated evaluation of group membership.

The measurements of pre-parsed query execution times show visible overhead of evaluating the access rights. We can see that the overhead stays within 200 % over the baseline for every user except the last percentile. And even then, the maximum measured overhead is 400 % more than the baseline of searching

without the access rights. For frequent words, however, there is a negative overhead of the access rights evaluation, because of restricting the documents to evaluate to those accessible by the user. The reason of such behaviour is that decoding of long list of document IDs is fast enough (and it can be made even faster using skip-lists), and what does matter is how many times the decoding has to be interrupted in order to look at another list (for another word/virtual token).

We expect the overhead will be even lower for phrase and multi-word queries, which are prevalent: their cost is already much higher because of evaluating the positions of words, and the access rights evaluation time would remain about the same.

7 CONCLUSIONS

We have proposed a novel method for handling the access rights within an intranet full-text search engine by extending a general-purpose search engine with additional support for virtual tokens, which can be processed more efficiently than ordinary words during the query execution. This extension can be used for many other purposes apart from access rights encoding (such as social search, date/time limiting, subtree limiting, per-agenda search, etc.).

Many systems of the access rights can be translated to the search engine data as virtual tokens representing groups of users or individual users. There are some cases where this representation is not straightforward, such as a set complement of the group, or intersection of two arbitrary groups.

We have implemented the proposed system, indexing static documents and dynamically-generated content of a large intranet system with a complex system of access rights and rules. The measurements proved that the overhead of evaluating the access rights for a given data set is under 200 % for 99 % of users, and at most 400 % for the worst case.

The main drawback of the proposed method is that caching of the access rights of the documents is an integral part of the system, so the permissions cannot be revoked except by reindexing the document again or marking it as deleted. The caching of per-user access rights is, on the other hand, purely optional.

Further improvements are possible: the most promising one is to use skip-lists for large lists of document IDs.

ACKNOWLEDGEMENTS

The authors would like to thank Pavel Šmerk and Mirka Kramáreková for careful proofreading of this paper.

REFERENCES

Anderson, R. J. (2008). *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing.

Bailey, P., Hawking, D., and Matson, B. (2006). Secure search in enterprise webs: tradeoffs in efficient implementation for document level security. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 493–502, New York, NY, USA. ACM.

Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE Trans. Inform. Theory*, pages 194–203.

GSA (2009). Google Search Appliance. http://code.google.com/ intl/en/apis/searchappliance/ documentation/.

Hawking, D. (2004). Challenges in enterprise search. In *ADC '04: Proceedings of the 15th Australasian database conference*, pages 15–24, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Hurley, J. (2009). Preventing information leakage in the search engine. Master's thesis, University of Tromsø.

IS MU (1999–2010). Masaryk University Information System. http://is.muni.cz/.

Judy (2002–2010). Judy Arrays Web Page. http://judy.sourceforge.net/.

SAML (2005). Security Assertion Markup Language. http://docs.oasis-open.org/security/saml/ /v2.0/saml-core-2.0-os.pdf.

Zhu, H., Raghavan, S., Vaithyanathan, S., and Löser, A. (2007). Navigating the intranet with high precision. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 491–500, New York, NY, USA. ACM.

Zobel, J. and Moffat, A. (2006). Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6.