Department of Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University, Bratislava

Author:
Anton Kohutovič

blog.matfyz.sk

community blog portal

Master thesis
Thesis advisor: RNDr. Martin Homola

Bratislava 2008

By this I declare that I wrote this diploma thesis by myself, only with the help of the referenced literature, under the careful supervision of my thesis advisor.

Bratislava, April 2008 Anton Kohutovič

Contents

Abstract

Weblog became a phenomenon with high socio-economic impact on the society. The number of blogs doubles every six months. Most of algorithms dedicated to ranking web documents are not ideal also for ranking blogs. Therefore, several blog specific algorithms were designed. Some of them use similar ideas such as the best known PageRank and other try to consider behavioural features of users.

XML is a language suitable for exchanging and storing data. It is a simple, easy to parse and also human readable plain text format. Data stored in XML are well structured, self-descriptive and unbound to any platform. The concept of usage of XML as a storage is not new, but applicable only with small portions of records. There is a lack of physical pointer which facilitates data querying. XML databases try to tackle this problem and they provide full featured storage engine where XML data can be queried and updated with help of effective indices.

This work tries to combine both researches, XML databases and ranking algorithms, into one project – a community blog portal `blog.matfyz.sk`*. The idea in the background was to develop a communication environment where people around Faculty of Mathematics, Physics and Informatics can present their works, opinions and suggestions.*

Keywords: weblogs, blogosphere, XML, XML native databases, ranking algorithms

Chapter 1

Introduction

1.1 Background

Social media has become a modern and popular way of communication and collaboration inside communities. People interact each other by words, pictures, videos and audio recordings. Internet provides a broad spectrum of virtual communication platforms. Forums, message boards, weblogs, wikis, podcasts, voice over IP, virtual worlds (Second life) and social networks (MySpace, Twitter) and other services which enable creation of communities.

1.2 Motivation

Weblogs belong to the earliest social media and they are still very popular and their influence is permanently growing. We set up a goal to develop a blog portal for the community of the Faculty of Mathematics, Physics and Informatics (FMFI). There are many implemented open-source blogging tools and therefore implementing a blog itself is not a challenge. The real challenge is to build up an experimental blog portal, where many interesting approaches can be examined and proved. Social networks propose a great amount of research problems: searching and indexing objects, classification of objects, evaluation and ranking of objects, tracking threads, collaborative filtering and spam identifying etc. We focused on the evaluation and the object ranking algorithms, but our portal can be also used for further research topics.

Another challenge was to construct our application on a non-relational storage engine and to examine what possibilities and drawbacks it gives. A concept of usage XML for data storing is several years old, but there is still a lack of real application using XML databases.

1.3 Related works

The first important study about weblogs ranking was performed by Adar et al. [AZAL04]. They suggested a nontraditional ranking system *iRank* based on the implicit links structure of weblogs. iRank assigns high rating to the sites that contain original information in contrast to PageRank [PBMW98] and HITS [Kle99] which better evaluate popular pages. The main idea of the PageRank algorithm is used in two approaches: BlogRank and B2Rank. BlogRank was designed in 2006 by Kritikopoulos et al. [KSV06] at University of Economics and Business in Athens. Implicit links, based on similarity of topics, are also taken into account by construction of entries graph and this is the main difference to PageRank. Graph of entries enhanced by implicit links is used also for computation of the latest ranking algorithm we have studied – B2Rank from year 2007 [THM07]. This algorithm tried to include users' behavioural features in the computation and the authors tried it quite successfully on the Persian blogosphere. Also, Fujimura et al. propose a new algorithm called EigenRumor based on eigenvector calculations [FIS05].

1.4 Results

We employed native XML databases in order to build up a blog portal. Our work contribute to detect important errors and drawbacks of selected projects. We also designed and implemented adjustments of the interesting algorithm EigenRumor. These changes improve results of the algorithm in such community, where additional information (votes, comments) can be acquired.

1.5 Thesis outline

Our work is structured into 6 chapters. We summarised knowledge about XML databases in the Chapter 2. At first, we made a brief introduction to this concept (section 2.2) and then we presented our experiences and conclusions about selected projects (section 2.3). Chapter 3 offers an overview to weblogs and their impact to the society.

We collected and studied almost a complete set of nontrivial ranking algorithms designed for blogs, which were presented at various conferences about blogging and Web. Many of them have the same principles as *Impact factor*, the measure of the citations to science and social science journals. We selected and studied more rigorously one of these ranking algorithms. In the section 4.2.4, we proposed our adjustments and suggestions to that algorithm. Our implementation is described in more details in the Chapter 5 and the following Chapter 6 consist of our results and conclusions that we have done in our work.

Chapter 2

XML databases

2.1 Brief introduction to XML

The Extensible Markup Language (XML) is a language derived from a simplified subset of SGML and primary designed for sharing and exchanging data across different information systems. The main features of the XML are the simplicity, and the flexibility. The XML is a type of a meta language easily extensible for domain specific languages. The XML do not have any predefined tags as HTML and also its syntax is stricter. It is focused on the data description, not as HTML on data presentation. The XML was designed to be an open format unbounded to any platform. It is ideal for storing semi-structured data, and because most information have some non-linear structure, it is suitable for many modern applications.

A XML element consists of a logical and a physical structure. The physical structure is built up from units – entities. An entity may refer to other entities to cause their inclusion in the document. Each XML document has only one "root" element, which is the parent of all other elements. The logical structure is built up from declarations, elements, comments, character references and processing instructions, which are indicated in the document by explicit markup. [W3C06]

A XML element starts with the starting tag and is enclosed by the ending tag. Each element can have some attributes, which express element's properties.

There are two requirements on XML documents. They must be well-formed and valid.

1. **Well-formed** documents must satisfy all XML's syntax rules. Each element must have an opening and a closing tag. Elements cannot overlap.

2. **A valid** document must include DTD or XML schema definitions. All tags must be defined.

Listing 2.1 Example of a XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rootElement SYSTEM "example.dtd">
<rootElement>
  <!-- This is a comment-->
  <nestedElement>
    sample of PCDATA string with &amp; entity
  </nestedElement>
  <nestedEmptyElement />
  <!--
    abbreviation for:
    <nestedEmptyElement></nestedEmptyElement>
  -->
  <elementWithAttributes value="10"
                         status="published" />
</rootElement>
```

The XML became a recommended standard by W3C in 1997. Members of this organisation are also important companies in IT business (Oracle, IBM, Sun and Microsoft), therefore the XML reflects requirements of real business.

Advantages of XML

- The XML is simple language. It is well legible for humans and machines.
- It supports Unicode.
- The hierarchical structure of XML documents can fit for many types of documents
- Hence of the strict syntax, it is very simple to parse XML documents. It is a context-free language.
- It is easily extensible through namespaces.
- Standardised

Drawbacks of XML

- The requirement of start and end tags for delimiting data gives a large volume overhead.
- The XML processing is slow due to the verbosity and the lack of physical pointers. [Bou05]

2.1.1 Usage of the XML

The XML simplifies data sharing and interchanging in the world where each database uses its own format, incompatible to each other. Because of plain text format of XML, every application can process it. Almost all languages used in service orientated architectures are derived from the XML. It is the alphabet for all web languages e.g. XHTML the version of HTML, WSDL for describing available web services, WAP and WML as markup languages for handheld devices, RSS languages for news feeds, RDF and OWL for describing resources and ontology, SMIL for describing multimedia for the web. The future of the Web will be strongly bounded with the XML. The service orientated paradigm slightly fulfils holes upraised by increasing requirements on development speed and flexibility.

2.2 Definition of XML databases

One of reasons for using XML databases is increasing common use of XML for data transport. Data are often extracted from databases and transformed into XML documents and vice versa. Probably, XML database can eliminate the cost of transformation. XML database is a quite new concept on field where relational databases safely dominate. But we think, that it is worth to look at them closer, and to try which properties of XML database are ready for real projects.
In formal definition by XML:DB consortium states that a XML database:

- Defines a (logical) model for an XML document - as opposed to the data in that document - and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models include the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0.
- Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.
- Need not to have any particular underlying physical storage model. For example, Native XML database (NXD) can use relational, hierarchical, or object-oriented database structures, or a proprietary storage format (such as indexed, compressed files).

2.2.1 Tables versus collections

"When your only tool is a hammer, everything looks like a nail. When your only tool is a relation database, everything looks like a

table." [Har05]

The world around us is not simple. We can try to describe reality with tabular data, but why should we always use tabular data, if we have a more sophisticated tool to do it. A tree structure can fit closer to reality than n-ary relations and the XML has all features to success in this domain. Many people believe that it will dominate the Web in the future.

Storing XML documents in ordinary files may be used only for small amount of data. Processing a large set of XML documents in files is very time consuming operation. Better solution is to store XML in a database which provides smart and fast indexes and supports general querying languages. There are two types of XML databases. XML-enabled database (XEDB) are built on top of relational databases and XML data is mapped to a relational database but native XML databases (NXDs) are intend especially to store XML document in their native structure.

A main unit of a relational concept is a table, where records are stored in rows. An equivalent for rows in NXDs are documents. Documents are organised in collections; equivalent of tables. Collections are usually associated with some schema, which defines the data model for collections. In this case, all documents must satisfy one common schema. There are also NXDs, that allows collections without any specific schema – schema-independent NXDs. Documents in a collection can vary in their structure, but it is still possible to construct queries across all documents in the collection.

Schema-independent collections give to a database a lot of flexibility. It is easy to change a structure of documents later during the development or even after the deployment. Applications built upon such collections can be developed faster, but this approach bring risk of low data integrity [Sta01]. The application, we have implemented, uses these schema-independent collections. It was important to have our project as flexible as possible.

In generally, NXDs do not have any common storage structure. They can use relational, hierarchical, or object-oriented database structures and also a storage in a file system. Many implementations uses own proprietary format.

2.2.2 Querying

Databases are places dedicated to storing and effective retrieving information. Our world is dependent on databases much more than we admit. The most common way for querying a database is SQL language. The database will interpreter the SQL query, evaluate all expressions and select requested data in tabular structure. Native language for querying on NXDs is XPath. XPath is a language developed for addressing portions of a XML document, though it is also ideal for queries. The XPath is language only for data reading, so

a database only with XPath would be read-only and semi-functional storage. There are another languages built for purpose of updating: XQuery, XQL, XUpdate.

XPath and the XQuery are based on path expressions to navigate through the XML hierarchical structure, which is modelled as an ordered tree. For example a query `/user//post/title` will select all title nodes with parent node post and with ancestor node user. The single slash express parent-child relationship and double slash ancestor-descendant relationship. XPath defines additional relationships by axis specifier. Preceding-sibling and following-sibling relations also belong to supported axis. XPath can reduce a result set of nodes by using predicates. Predicate expressions are enclosed by square brackets. E.g.: `/user[@ID="1"]//post/title[contains(.,"blog")]` will select all titles which contains word "blog" and they are descendants of user node with attribute ID equal to 1.

XQuery is a powerful convenient language designed for processing XML data. It implements XPath as language for addressing but it have some similarities to SQL. Each XQuery program is an expression which is evaluated to a value. "1+1*5" is the valid XQuery program with a return value 6. There are five basic keywords used in XQuery, commonly abbreviated as FLOWR.

FOR *variable* in *expression* – make a loop over elements of a sequence. This sequence can be defined by range (e.g "1 to 10") or by XPath expression.

LET – define local variables.

WHERE *expression* – filter result set according to an express.

ORDER BY *expression* – sort result set of node by XPath expression.

RETURN – return data.

There is also one conditional expressions IF. It must have both, then and else clauses, defined. `IF (10 < 1) then "true" else "false"`

Example We will show an example how to select all users, which have an attribute *type* set to *student*. We will count their posts and select only those with more than 5 posts. We will sort then these users by their registration date (element since). And finally, we will return result in format we need.

```
FOR $user in /user[@type='student']
  LET $postCount:= count($user//post)
  WHERE $count>5
  ORDER BY $user/info/since
RETURN <student postCount="{$count}">$user/*</student>
```

XQuery has own update extension, used for data manipulation. Kimbro Staken in [Sta01] marked update as most weakness point of NXDs. Situation has slightly changed since 2001. Many products offers good implementation of a XUpdate [1] and it is not necessary now to get whole document, make transformation and put it back. The XQuery update, and the XUpdate works now fine in some implementations and there are ready for using. Although we have experienced troubles with updates especially with project eXist.

indexes

Another interesting topics for each database are indexes and performance. Early NXDs did not support indexes so the performance results was very poor. Again, situation has changed and we claim that, our application based on NXD reaches satisfactory response times. Current XML databases support structural, fulltext, range indexes[2]. It is enough to do a fast and usable application. Although, we have made certain of that it is not always simple, or flawless. More technical information about implementations of indexes can be found in [Mei02] and [FGK06].

2.3 Implementations

There are many implementations at this time, but only a few of them are still in development and fresh. The following XML databases are known to provide an implementation of the XML:DB API defined by the XML:DB Initiative: eXist, Apache XIndice, Sedna, OZONE, DOMSafeXML. We tried eXist, Sedna and one else, Berkeley DB XML.

Native XML database was a good choice for our project. It is a relatively new technology worth for experiments. We wanted to examine how mature this technology is and if it is suitable for using in complex applications, such as our blog portal. We selected most active projects of the year 2006 and then we reduced the selection to three most perspective. The first attempt we made with Berkeley DB XML. When we decided to try a native XML database for our project, we had to find some appropriate implementation, which would satisfy our requirements.

[1] XUpdate is a simple XML update language. You can use it to modify XML content by simply declaring what changes should be made in an XML syntax.

[2] Structural indexes contain information on location of element and attributes. Range or value indexes store information of elements and attributes values.

2.3.1 Oracle Berkeley DB XML

The Berkeley DB XML was the first of databases which we wanted to try. This project had a long life and it was bought by Oracle, what promised good results. It is now developed as an open-source project, but it was not so in that time, what was a first disadvantage of this embeddable XML database with XQuery-based access to documents stored in containers and indexed based on their content. Current XQTS[3] score is 99.5%. The database is built on top of Oracle Berkeley DB and inherits its rich features and attributes. There is no need to do human administration, all is left to applications. All features of Oracle Berkeley DB such as transactions (ACID transactions/recovery) and replication are inherited. Both types of collections – schema-less and schema-constrained are supported.

We made some test to benchmark its performance, but we founded it not satisfactory as we had with other two solutions. We redone a benchmark test in January 2008, and performance significantly improved. Unfortunately, Berkeley DB XML does not support any update language as XUpdate, or XQuery update. Modifications can be executed only by changing a content of a node. There is no tool to do more complicated updates. So we would have to write our own interface for update purposes and then our application would not use any standardised approach for manipulation with the XML and that's why it would be impossible to replace Berkeley DB XML with another NXD. We would recommend this embeddable database to applications mostly with read only requirements. It can be successfully used with small resource overhead because it is file based and do not follow client/server architecture.

2.3.2 eXist

The next native database we tried was eXist developed by team leaded by Wolfgang Meier. It is an open source project with active development and it is distributed under GNU LGPL license. There is necessary a Java virtual machine to run this application and hence this fact, eXist is a platform independent database. It is not a problem to deploy it on Linux/Unix, Windows, Mac OS or any other operating system where the JVM runs.

The first advantage of this application is a very simple and user friendly administration interface. There is a quite useful GUI which can be used to all administration and maintenance tasks. Of course, there is also a command-line interface and it is possible to do with it the most of tasks, which can be done with GUI, but not all of them. We think, that CLI should be a more complex tool than a GUI, it is more useful for applications developers to execute any

[3] "The XML Query Test Suite (XQTS) provides a set of metrics for determining whether the W3C XML Query Language can be implemented interoperably as published. XQTS will help implementers identify possible problems both with the Specification and with their software." [W3C07]

operation on a database with a shell script rather than do it by clicking in GUI. We hope it will be improved in next versions.

eXist supports many web technology standards and languages for XML manipulation:

- XQuery 1.0 / XPath 2.0
- XSLT 1.0 (using Apache Xalan) or XSLT 2.0 (optional using Saxon)
- HTTP interfaces: REST, WebDAV, SOAP, XMLRPC, Atom Publishing Protocol
- XML database specific: XMLDB, XQJ/JSR-225 (under development), XUpdate, XQuery update extensions (to be aligned with the new XQuery Update Facility 1.0

eXist is now highly compliant with the XQuery standard. Current XQTS score is 99.4%. The query engine is highly extensible and features a large collection of XQuery Function Modules.

Another interesting feature is a support for the XInclude. The XInclude is a mechanism which allows to insert a XML chunk into any XML node without need of data replication. There is used XPointer as system for addressing XML chunks of included data. XInclude can help to avoid using of redundant data which makes updates more diffcult. It also changes tree structure of the XML to an undirected graph. Other databases we have examined do not implement this feature.

eXist can be integrated with a lot of types of applications; it can run as a stand-alone server, as a servlet or it can be embedded directly in an application. XML documents are stored in schema-less hierarchical collections. Query engine of this database implements efficient index-based query processing.

Developers of eXist have implemented an auto-index feature, what is a thing that makes queries faster without laborious manual creation of all appropriate indexes (but manual indexes are still necessary and should be used) but on the other hand the current implementation of auto-index mechanism is quite buggy. We had to deal with this uncomfortable property many times and we were forced to solve it with not only trivial and easy methods. We observered that most of problems occurred when a lot of update queries were executed in a short time. Mainly by high server load, for example by computation of ranking algorithm we have implemented (We will describe its details in Chapter 4). Large updates caused inconsistency of indexes and the result was, that a lot of other queries were incorrectly evaluated. E.g. when we requested all available posts with `//post`, we got not only post nodes in the result but some comment nodes, which logically should not be there.

Another symptom of corrupted indexes was that sometimes some nodes seemed to be invisible for queries although they were physically in the database

storage. Update queries, which used incorrect select queries, inserted or modified nodes which were marked by wrong index. This problem often violated smooth run of our blog portal and it needed manual interventions from us. In most cases, it was enough to execute a manual re-index of all indexes in our collections, what did not take a lot of time because of small size of the database. We also programmed a shell script timed by cron deamon, which every hour did the manual re-index as prevention. Unfortunately, there was ca. two times a much more damaged database. Not only indexes but also a file where names of all elements were kept. In these cases, the database seemed to be empty for each external or internal query. We contacted developers of eXist, and they suggested only one solution: to delete all data physically from storage and then to restore them back. This drastic solution was not so painful as it sounds only because we could export all data from databases in this condition.

We reported all problems we experienced to eXist team, but they were not able to help us, even after longer negotiations, where we offered them all access permissions, logs and data. They could not predict what causes problems without exact sequence of instructions, which deterministically damage databases. It is almost impossible to find out such sequence. We can log all update requests, but we cannot identify a point in the time, when something has started go wrong. We recommend them seriously to focus on this problem, because this problem is critical even with not huge portions of XML data. We can handle it only thanks to fact, out portal is an experimental project, but it seemed that eXist has more bugs than is acceptable in a business applications. It is very hopeful project and if developers fix all these critical errors it will be one of most interesting XML native databases on the market. . eXist gave us the best performance result at the end of 2006, despite it is implemented in java, which is not the most suitable programming language to write very fast applications with effective memory management. At the beginning of 2008, eXist had worst average performance results. Sedna and also Berkeley DB XML defeated eXist in most queries. But eXist had a domination in a one type of queries. All queries with ancestor-descendant relationship (e.g.: `//post`) had the same duration as queries where complete path of nodes were specified (e.g.: `/user/blog/post`).

We tried two HTTP interfaces, SOAP and REST[4]. REST interface was faster than SOAP for queries with a small result and slower for large set. It is because SOAP has an overhead in communication such as complete SOAP envelop, where REST uses only the simple GET HTTP method.

There are four files with indexes: words.dbx, dom.dbx, elements.dbx and collections.dbx. All indexes are organised in B^+-trees. The indexes are created

[4]REST (Representational State Transfer) is a software architecture designed for distributed hypermedia systems such as the World Wide Web. It was introduced in 2000 in the dissertation of Roy Fielding. REST is based on HTTP and uses it's methods GET, PUT, POST, DELETE to access and manipulate data.

for a whole collection, not for documents separately. This helps to keep inner B^+-trees pages quite small and it improves performance of queries to entire collections. More information about implementation details is published in [Mei02].

2.3.3 Sedna

The third project, we tried, was Sedna. This XML database system is being developed by the MODIS team at the Institute of System Programming of the Russian Academy of Sciences. It is written in Scheme and C++. Static query analysis and optimisation is in Scheme and other parts as parser, query executor, memory and transactions management in C++. As a basis for Sedna's data model was taken XQuery 1.0 language specified in [BCFF03]. In order to support updates they used the XQuery update extension, similar to eXist but closer to a syntax described in [Leh01].

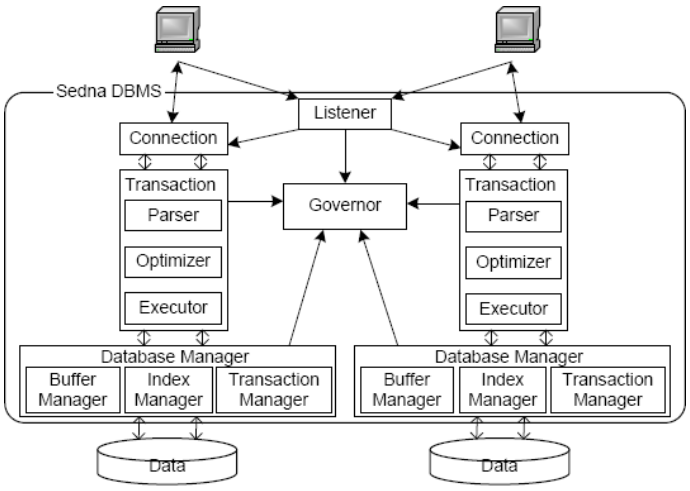

Figure 2.1: Sedna architecture according to [FGK06]

Main component of Sedna is *governor*; a control centre of the system. *Governor* knows the running databases and transactions. All databases runs separately and crash of a database would not affect other databases if the *governor* is running. The *parser* transforms a query to it's logical representation. This representation is used by *optimiser*. Optimiser creates an optimised execution plan which is a tree of low-level operations over physical data structures [FGK06]. Then, the execution plan is interpreted by *query executor*.

The eXist has good support for full-text search, unfortunatelly, sedna not. DtSearch, external commercial fulltext engine can be integrated with Sedna

but native support is still missing.

We tested Sedna 1.x and later also Sedna 2.x. Performance results with version 1.x was worse than contemporary eXist's results and therefore we decided to develop our blog portal with eXist. In January 2008, after mentioned problems with indexes consistency in eXist, we began test on Sedna 2.2. We have quickly rewritten all read-only queries, now compliant with Sedna. Update queries in Sedna and eXist have slightly different syntax, but they are also easy convertible. The performance results (see 2.3.4) of Sedna 2.2 was also interesting, so we did complete migration from eXist to Sedna in March 2008. Unfortunately, there were an issues that we had to solve. Some of our update queries caused always crash of the database and also database *governor*. Of course, we reported this to MODIS team. Their answer was incredibly fast, and a new built of Sedna where this bug was fixed, was released the next day.

We have deployed Sedna on out production server in March 2008 with high expectations of improvements in the performance and mainly in the stability. We have found out that Sedna crashes from time to time. Average 2 times in week. We identified this as a bug with insertion of strings in database. MODIS team considered this to be a serious problem and they immediately started to work on this issue after our detailed report.

In the April 2008, Sedna 3 was released which should repair mentioned stability issue. We have not observered more problems for now although Sedna is not well prepared for 64 bit architectures.

Sedna team have implemented a great feature SQL Connection, not commonly present in NXDs. SQL Connection allows access to relational databases in XQuery queries using SQL. Results of SQL queries are relations represented on-the-fly as sequences of XML elements representing rows. These elements have sub-elements corresponding with the columns returned by the SQL query and thus can be easy processed in XQuery. Connection is established via ODBC drivers. Supported are direct queries and updates on a databases, such as prepared statements and transactions. [MT07]

Following example demonstrates the use of SQL connection in XQuery:

```
declare namespace sql="http://modis.ispras.ru/Sedna/SQL";
let $connection :=
   sql:connect("odbc:MySQL ODBC 3.51 Driver://localhost/weblog",
               "userName", "password")
return
   sql:execute($connection, "SELECT * FROM provisioningMatrix
                             WHERE postID = 'p56'");
```

We will get a set o tuples as the result:

```
<tuple postId="p56" userID="u13"/>
```

This SQL connection is a helpful tool for developers. The relational and XML databases can be integrated in one application at database tier without application level modifications. Some data is better to store in relational databases and some in XML database. Both database concepts have own advantages and a compound application can profits from benefits of both and avoid handicaps of each concept.

2.3.4 Benchmark tests

We made a program for a comparison of a performance of the databases. It measures an evaluation time for each query and each database. This program sends queries in short random intervals and computes average time of selected query. The tests were done at the same machine in the same time and with the same data to ensure identical conditions. We selected different types of queries, evaluation of each is based on a different principle.

Tested queries

1. ```
   subsequence(
   (let $list :=
       (
         for $current in collection("weblog")/user/blog/post
           [@status="published"]
           [@accessCount>50]
           [contains("sk",@lang)]
           order by
             xs:integer($current/@accessCount) descending
         return $current
       )
       for $current in $list
         let $user := $current/ancestor::user
         return
           <post>
             {$current/@*}
             <user>
               {$user/@ID}{$user/@type}
               {$user/info/*}
             </user>
             {$current/title}
             {$current/subtitle}
           </post>
   ),1,10)
   ```

2. ```
   /user/blog/post[@ID="p71"]
   ```

3. `//post[@ID="p71"]`

4. For eXist:
   ```
   let $ac:=/user/blog/post[@ID="p71"]/@accessCount
   return update replace $ac with $ac+1
   ```
   Equivalent for Sedna:

   ```
   update replace $i in /user/blog/post[@ID="p71"]/@accessCount
   with attribute accessCount {$i+1}
   ```

   Because of lack of any complex update language in Berkeley DB XML, we did not test it for performance of modifications.

5. ```
   update insert
   <comment>
     <title>test</title>
     <content>Testing content</content>
   </comment>
   into /user/blog/post[@ID="p71"]/comments
   ```

As we can see in the table 2.1, Sedna has the best results for the most complicated query, but results for query with double slash are not so impressive, almost unusable. The second value in the Sedna's column is query time after addition of an suitable index. In the current version of Sedna, query executor does not use indexes automatically so we must to construct queries, which do not conforms XQuery specification. We decided to use indexes in Sedna only for testing, because we want to have our application ready for use with eXist as well as Sedna or any other native XML database with correct XQuery support. But Sedna's results with indexes look very promising, we hope that indexes will be used by the query executor soon and without necessary modifications of queries.

The results in the test query #5 strongly depends on the results from query #2. Updates are so fast as ordinary reading query, only no data is transmissed.

We have got a more worth results from logs we have collected in the production conditions with real users and real requests. We used only eXist and Sedna so we know only their values. The average page generation time with eXist database was 1.33 second. It so high because of long responses when almost a whole heap was used and until garbage collector was cleaning it. Median of these times was notable smaller. Only ca. 0.37 second. Sedna achieved better average result, but it has slightly worse median. Average page generation time: 0.78 second. Median: 0.41 second.
   ```

Query	eXist SOAP	eXist REST	Sedna	Berkeley DB XML
#1	92.6	102.7	25.9	44.6
#2	8.2	3.5	6.7/3.9	19.8
#2 – more data	41.5	30.7	7.5/4	17.7
#3	7	3.6	292.3/4	108.5
#3 – more data	42.5	34.8	415/4.1	101.6
#4	6.7	4.6	15	-
#5	7.2	5.2	14.5	-

Table 2.1: Average times for queries 1-5 in milliseconds.

Chapter 3

Blogosphere

> *"A blog (a portmanteau of web log) is a website where entries are commonly displayed in reverse chronological order. "Blog" can also be used as a verb, meaning to maintain or add content to a blog.*
>
> *Many blogs provide commentary or news on a particular subject; others function as more personal on-line diaries. A typical blog combines text, images, and links to other blogs, web pages, and other media related to its topic. The ability for readers to leave comments in an interactive format is an important part of many blogs."* Definition from Wikipedia, March 2008

> *"Blog is short for weblog. A weblog is a journal that is frequently updated and intended for general public consumption. Blogs generally represent the personality of the author or the Web site. Blogs have common elements: updated frequently (usually daily); informal; grouped by date with links to archives of older posts; informative and/or inspiring (the good ones); frequently linked to other sites that inspired the blog; and addictive for those who blog."* Definition from BloggerForum.com

3.1 Global social phenomenon

Everybody can now express its own opinions, feelings or describe his experiences through his personal weblog. The concept of a diary is old and many people used to write it. Probably for transformations of moods and thoughts to the written form or only for recording things that everyday life brings. Blogs extend the possibilities of diaries now. It is still very personal and in some cases almost intimate but the audience and the way of publishing has changed. It is not necessary now to write a book or to find a publisher of news to acquire readers. There cannot be more simple way how to share with the ideas with

people than to write a blog. It is cheaper than non-electronically publishing and it also offers more possibilities. Although blogs have primary textual form, there are several derivations: photoblogs, vlogs (video blogs), music blog (MP3), audio (podcasting). Blogs have usually a space for readers' reactions and comments. It makes blogs a two-way communication medium. Convectional journalistic forms as an article in newspapers, a radio and television transmission are only one-way message. [Tor05]

The blogging became a world spread social phenomenon which impacts public opinion, politics, business and other aspects of public and private life, even the way how people communicate with their families and friends. Blogosphere is one of the most expanding parts of the Web. A weblog tracking and searching portal Technorati collected till March 2008 about 112.8 million links to weblogs. Some of medial analyst consider blogs to be "Next internet boom" and they predict a huge investment into this small publishing space with high potential. Blogs have several features which give them important advantages.

Firstly, people like to read such texts where they feel a personal attitude of an author. They want this personal contact because the most of email in work are too formal. It is easier to read an opinion if we can imagine somebody who writes it and we know his face from photograph aside the article. We know his opinions because we have read his recent postings and we know, he will publish new texts in the near future and we know what we can expect. Bloggers also have own favourite blogs and bloggers and they will certainly link them. And so, a community is set up.

Another advantage is a great position of blogs in search engines. There are several factors which influence ranks in engines such as Google. One of them is up-to-date content. Blogs are regularly and often updated pages. If a blogger wants to attract new readers and retain the old ones, he has to permanently update his page. Reader can also change content. In the most of blogs, reader can react to articles and they are allowed to add comment and opinions. Bloggers usually create many links to other interesting blog, or other places worth to visit. They can also react to foreign article with their own article. It is possible to find the whole topic related threads with many posts replying to each other. These threads are often well interlinked. The count of inlinks is a fundamental measure for quality evaluation (See Chapter 4). This fact strongly improves results in search engines and good optimisation for search engines (SEO) is almost the most important step to success on the Web.

Blogs have a clear architecture focused on the content with well organised

links. Every blog is managed by the user[1] and therefore the content is full of relevant keywords. It is proven also in our experiences. We wrote a short, almost contentless post about possibility of including a video from YouTube for our users. Keyword "youtube" is lucrative and therefore we were surprised that our post was in top 10 results in Slovak language for some time. It has about 25,000 visits but there are almost no links to it.

The creation and using of a blog is simple. Everybody can publish and manage his own personal blog without any specific technological know-how. There are many open-source solutions for blogging which can be deployed on the most of free web hosting. All solutions provide user-friendly interface for writing a new article, managing photo and video galleries. There are also many companies providing blogging services with their software. (e.g. `blogger.com`, `blog.sme.sk`). Most of them are free, but there are several paid services providing a special functionality.

3.1.1 Blogs & business

Most of blogging people do it as a hobby, or as a way how to share thoughts and viewpoints with friends or wider group on whatever topic. Blogs are also used as photo galleries from holidays or journeys with some commentaries. It is better to inform friends about past experiences in a one post with photos rather than to send them emails with attached images.

There are some bloggers who wants to turn their blogs into a successful business. Some blogs have already succeeded. The Coudal (`http://www.coudal.com`) and SimpleBits (`http://simplebits.com`) are good examples of blogs that initially focused on content, but now, they sell advertisement space and products. As blogging matures, there are more and more people who are quitting their jobs and starting to blog full-time [BB06]. According to an older work [Tor05], in a 12 month period, 45% of blogs do not generate any revenue. About 40% earn under $5,000 and about 4% generate revenue over $100,000.

Many companies and business organisations have recently discovered that blogging can help them to reach their internal and external communication goals. Business blogs differ mainly in their purpose and the auditorium. We recognise 3 main categories of business blogs [BB06].

Company blog presents a company and its business in general. It works like a propagation brochure with newest announcements. A company blog can replace a typical home page of a company.

[1] Of course, one user can have more blogs, or a group of users can write their blog so-called "Collaborative blog"

Product blog propagates certain products or services and it is built for promotion of sales.

Brand blog is used to communicate marketing messages. It helps to extent company into new markets. Boeing, Microsoft and Sun keep their own brand blogs. Some brand blog can be sponsored by a company but written by person outside the company.

3.1.2 Blogging in numbers

The Blogosphere is an enormously growing part of Web. According to Technorati statistic [Sif08] from report in April 2007, there are about 120,000 new blog every day, 1.5 million posts per day. Bloggers blog mainly in Japanese (37%). The second language is English (33%), then Chinese (8%), Italian (3%) and Spanish (3%).

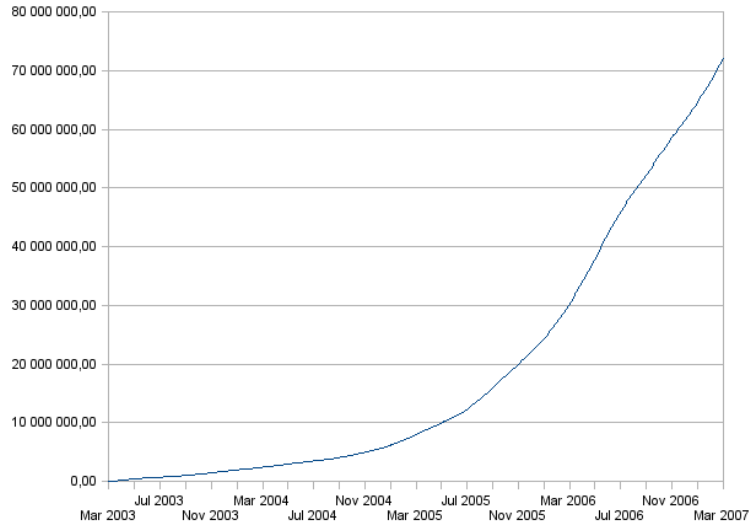

Figure 3.1: Growth of blogs' count in the Blogosphere from March 2003 to March 2007

From an older study [Rai05] made by Pew Internet & American Life Project, there are 7% of adults in United States which have created a blog and 27% of them reads blogs regularly. The interactive feature of many blogs is reflected by 12% of internet users who have posted comments or other material on blogs.

3.2 History

Short summary of important events in the evolution of the blogs collected from various sources:

1990 – The beginnings of weblogs can be traced to this time Usenet and in it's Moderated Newsgroups. Most of them were only moderated discussion forums but there was a one exception. *mod.ber* was created and managed by only one person, Brian E. Redman. He stared to write there summaries of interesting postings and threads on the net.

1992 – A site built by Tim Berners-Lee[2] at CERN. He points to new interesting sites which came on-line.

1993, June – NCSA starts to publish their "What's new" site, immediately followed by Netscape and their "What's new". These sites were lists of new sites which became on-line. Netscape archives are still available at `http://wp.netscape.com/home/whatsnew/`.

1997, September – Slashdot launches their "News for nerds".

1997, December – Jorn Barger coins the term web log.

1998, November – Cameron Barret published the first list of blog sites on Camworld.

1999, The beginning – Peter Merholz coins the term blog after announcing he was going to pronounce web blogs as "wee-blog". This was then shorten to blog. Brigitte Eaton starts the first portal devoted to blogs wit about 50 listings.

1999, March – Brad Fitzpatrick, a well known blogger started LiveJournal.

1999, July – Pitas launches the first free tool dedicated to building own blogs also for user without technical know-how.

1999, August – Pyra releases Blogger which becomes the most popular web based blogging tool to date, and popularises blogging with mainstream internet users. Pyra was purchased by Google in February 2003 and `blogger.com` is now world best know blogging service.

2001 – Blogs becomes to be a phenomenon. Established schools of journalism begins researching blogging. Blogging is very close discipline to journalism. Blogs are penetrating to politics and many companies communicate with employees and customer through the blogs.

[2]Sir Tim Berners-Lee, father of the World Wide Web, director of W3C.

2003 – The beginning of the Iraq invasion provides a significant catalyst for blogging.

2004 – The word "blog" became the world of the year at Merriam-Webster Dictionary. (`http://www.merriam-webster.com`)

3.3 Anatomy of blogs

All blogs have very similar structure cognisable at first sight. Blogs are typically laid out in reverse chronological order. The most recent entry is listed first. The main textual entries on a page are called *posts*. The post has a fairly consistent format composed of a *headline*, a link to the main source or web page under discussion, a description of the material, *commentaries*, image or photo, *permalink*, or quotations from the original source. All these elements can be used in various combinations.

The *permalinks* (shorten permanent link) are unchanging links to the specific posts as it is located in the larger database that powers the organisation of blogs. If somebody wants to inform about certain post he will send a permalink to this post. Permalinks make it easier to share links because otherwise a general URL for a blog will bring up the entire blog rather than a specific entry, and the reader must go through all the blog posts to find the desired information.

Traceback, another common element on posts, is an area that allows the blogger and his visitors to see what type of impact his post has had in the larger blogging community via a program that tracks who links to which blog post on the own blog.

Blog rolls is a list of links to other blogs that the blogger likes or recommend to read. Typically situated in a *Sidebar* – additional column on the side of the blog's main page. Side also includes *About*; brief information about blogger, his interests and whatever the author wants.

When a post is pushed down and off a blog's main page, it is usually available through *Archives*. Most of blogs display links to older posts on a monthly basis in the sidebar.

The most comfortable way how to track changes of favourite blog is *RSS syndication*. RSS enables to get news from blogs without opening whole sites. If some post seems to be interesting from description in RSS feeder, it is simple accessible via attached *permalink*.

Figure 3.2: The Anatomy of a common blog

Chapter 4

Ranking algorithms

At the beginnings of Internet, many people thought, that it will be possible to effectively index and search all documents. Today, we know that it is almost impossible. Current Web consists of "tons" of documents and their amount increases faster than we can process and catalogue them. Therefore there is a need of classifying documents by a "quality", where the the "quality" can stands for various measures based on various approaches. But the goal is the same; to provide interesting, helpful content for user in specific topic he is searching for. In the blogosphere it's quite similar even more useful and with special additional requirements. Readers often want from blog portal a offer of article which are fresh, interesting and present similar opinions. For all that it upraise a space for personalised ranking algorithms. There is still a large space for adjusting following algorithms towards the personalisation. Now, we will discuss about general ranking strategies.

We will classify the blog ranking algorithms into several groups (for more complex classification see [FIS05]):

1. Subject of ranking
   (a) Blog entries
   (b) Bloggers
   (c) Articles referred to by blogs
   (d) Goods or services referred to by blogs
2. Semantics of ranking
   (a) Support of community
   (b) Trustworthiness
   (c) Freshness
   (d) Specific attributes e.g. funnines or usefulness

3. Types of evaluation used for ranking

   (a) Hyperlinks – trackback

   (b) Access count

   (c) Explicit votes

   (d) Comments

   (e) RSS subscritions

   (f) Natural language analysis (comments)

4.1 blog.sme.sk karma

A blog portal blog.sme.sk maintained by newspaper SME is the most successful blog portal in the Slovak Republic. Author of this portal implemented own ranking algorithm. They named it karma according to score introduced at a well known website Slashdot, but only names are similarm the systems of ranking are completely different.

Developers of blog.sme.sk used two charactertics for the ranking. Count of votes for a post and count of views of apost. The first implementation of their ranking algorithm was simple combination of mentioned values.

$$karma = \left(\frac{views}{400}\right) + \left(\frac{votes}{100views}\right)$$

We consider this solutions not to be optimal, because posts with few views achieve much higher karma than others. It is well shown at figure 4.1. And another problem is that karma is not limited and it can grows to very high values almost lineary. Fortunately, they have repaired this ranking equation. The new karma computation removes these flaws.

$$karma = 5\left(\frac{views}{300 + views}\right) + 45\left(\frac{votes}{300 + votes}\right)$$

This new formula is based on the fraction $\frac{x}{1+x}$. Values for this fraction rise from 0 to 1 for $x > 0$. A graph of this functions have hyperbola shape and so its growth is damped for large values of x. If we replace number 1 with another constant C we will set for which x will this functions will have value 0.5. It is because if x is equal to the constant C, then $\frac{C}{C+C} = 0.5$. In our case C is equal to 300. The range from 0 to 1 is too small, so we multiply this fraction by 45 in the case of the fraction with count of vote. In the case with the count of view, we use multiplication by 5.

A rank evaluated the user quality is also called karma, and it is computed as average from karmas of his last ranked posts not older than 70 days (only with $karma > 0$). The karma is assigned to post only if its count of views go over number 100. We consider this innovated *karma* formula much more better, but there is a space for improvements towards personalisation and community.

4.2 Eigenvector based algorithms

There are several ranking algorithms developed to evaluate a quality of Web pages. Effectiveness of most successful solutions as PageRank and HITS was proved in academic and also in industry sphere. This approach would be also useful in the blogosphere, but we are going to describe more suitable ways, how to rank blogs. We will summarise approaches to blog ranking. This topic is a subjects of a research in several teams and the more detailed results over the extent of our work can be found in [THM07], [KSV06], [FIS05] and [AZAL04].

We will focus now on ranking algorithms which use citations as a measure of quality and which are based on a computation of an eigenvector for some particular matrix, mostly for an adjacency matrix. We will use an orientated graph of nodes, where we will construct an edge from node A to node B, if A cites (refers or evaluate) to B.

In algorithm, we have implemented, we do not use count of hyperlinks to a page for expression opinions of people, because we have small dataset in our portal. It is also proved that the graph of weblog entries is not very well connected [KSV06]. The links are sparse and the use of algorithm such as PageRank would not be the best solution. Another good reason is, that we have implemented our portal ourself and we can access information (e.g. voting, putting comments) which are usually unavailable for blog tracking or searching engines. Therefore, we used explicit evaluations in our implementation of EigenRumor algorithm described closer in the section 4.2.4.

4.2.1 PageRank

The PageRank is most known query-independent link citation measure. It was developed by Page and Brin [PBMW98] with goal to improve goal the quality of web search engines and it is used by a Google search engine as a primary link recommendation scheme. The The PageRank assumes random surfer who starts browsing the web at any random page. He clicks to a randomly chosen link on the page and then he continues to another page. There is a non-zero probability of a jump to a random bookmark (some link independent from the current page) for each page, it is called damping factor d

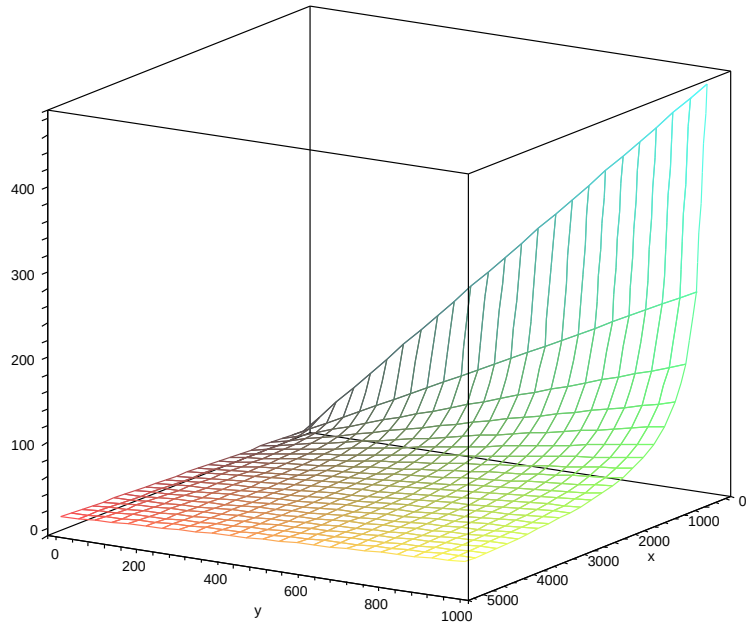

Figure 4.1: Graph of the old karma. $x = views$ and $y = votes$

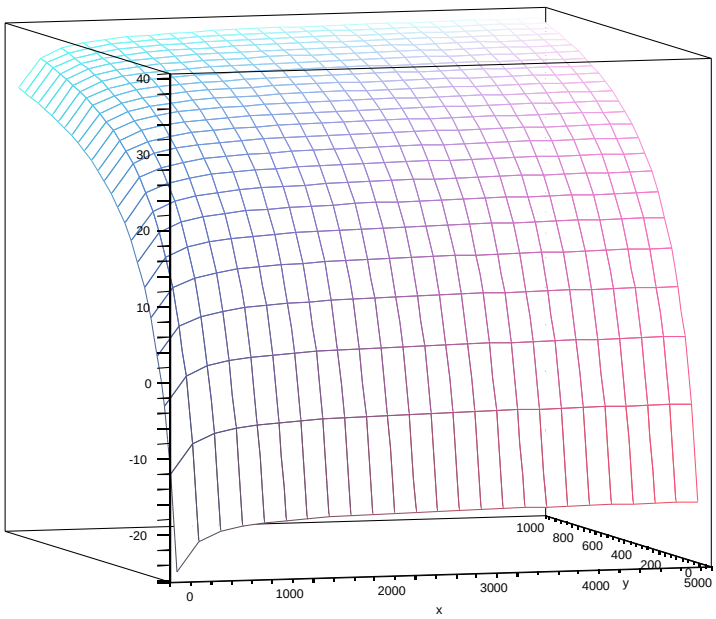

Figure 4.2: The distribution of the actual karma. $x = views$ and $y = votes$

Let page A have pages $T_1, \ldots, T_n$, that refers to A and parameter $d \in [0, 1]$) (damping factor usually $d = 0.85$). We define $C(A)$ as number of outgoing links from page A. Then PageRank PR for page A is defined:

$$PR(A) = (1 - d) + d\left(\frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_n)}{C(T_n)}\right) \tag{4.1}$$

The main idea of PageRank is that a page A, which has many links from other pages, has the high PageRank value the more that pages have higher PageRank. (See figure A.4) Values of PageRank are the values of the principal eigenvector $\vec{R}$ of the normalised adjacency matrix. [BP98]

$$\vec{R} = \begin{pmatrix} P_1 \\ \vdots \\ P_n \end{pmatrix}$$

We define adjacency function $l(p_i, p_j)$. If there is a no link from p_i to p_j, then $l(p_i, p_j) = 0$. $l(p_i, p_j) = 1$ if there is a one or several links from p_i to p_j Where $\vec{R}$ is computed by

$$\vec{R} = \begin{pmatrix} \frac{(1-n)}{d} \\ \vdots \\ \frac{(1-n)}{d} \end{pmatrix} + d \begin{pmatrix} l(p_1, p_1) & \ldots & l(p_1, p_n) \\ \vdots & \ddots & \vdots \\ l(p_n, p_1) & \ldots & l(p_n, p_n) \end{pmatrix}$$

We can compute $\vec{R}$ with iterative algorithm, it converge really fast. A few iterations must be done and it also gives good result in practise. The PageRank for 322 million link database converge in ca. 54 iterations, what is acceptable few and it seems that this algorithm scales very well also for extremely large collection. Deeper sight into mathemtical background of the PageRank are rigorously described in [LM04].

4.2.2 Hyperlink Induced Topic Search (HITS)

This algorithm was described by Jon Kleinberg in article [Kle99]. Main idea consists of partition a set of pages into two important sets. The hub pages and authority pages. Good hub page has to cite good authority pages, and good authority page has to be referred by good hub pages. It is mutually recursion definition, because it is needed to use second definition in order to express first. We define two scores for each examined page p. First $A(p)$ determines authority of page p. The second one $H(p)$ is the hub score, i.e. how is this page p a good hub for set of pages. The computation of the hub and the authority score is done as follows:

1. let V be the set of nodes in the graph of neighbourhood and

E is a set of all edges in the graph.

2. initialise $H(i)$ and $A(i)$ to 1 for all i in V.
3. while the vectors $\vec{H}$ an $\vec{A}$ have not converged do:
4. For all $i \in V, A(i) = \sum_{(i,j)\in E} H(j)$
5. For all $i \in V, H(i) = \sum_{(i,j)\in E} A(j)$

Convergence of the H and A vectors was proved by Kleinberg in. In practise, this vectors converge in about 10 iterations [BH98].

4.2.3 EigenRumor Algorithm

We are now going to introduce a generic algorithm suitable for ranking objects in the blogosphere but also applicable to any other web community. This algorithm was presented at workshop on the Weblogging Ecosystem and it was designed by Mr. Fujimura and Mr. Inoue from NTT Cyber Solutions Laboratories. We have implemented and adjusted their algorithm to use in our conditions and we are going to share our conclusion about it in following chapters.

Basic difference between EigenRumor and PageRank 4.2.1 or HITS 4.2.2 is in subjects of examination. The EigenRumor defines two entities: *agent* and *object*. *Agent* is represented by human being – blogger. *Objects* are used to represent any blog entity such as blog post. Algorithms 4.2.1 and 4.2.2 consider only *objects*, but these solutions can not be used in ranking of blogs with best results, because each blog post have its author, who has probably already post other *objects* influencing his popularity. Therefore, some new post of a popular author should have higher rank, however there are no links to it yet. We will use similar concept as in 4.2.2 but both score will be assigned to agents, not to objects. Objects have assigned Reputation score.

Community model

We assume a community of m agents and n objects. Each object is associated to an agent who published it. When *agent* i provide *object* j, we will make a *provisioning link* between i and j. Also each *agent* i can evaluate *object* j and so make *evaluation link* from i to j. Evaluation can be done in different ways. We can consider binary evaluation, if *agent* i cites or comment *object* m. Or *agent* i can assign an explicit score from some range. Now, we can construct provisioning matrix P, where $P = [p_{ij}]$ for $i = 1 \ldots n, j = 1 \ldots m$ and $p_{ij} = 1$ if there is provisioning link between i and j. Construction of matrix E is similar to construction of matrix P, but we can set value of e_{ij} to any real number from range [0,1] according to system of evaluation.

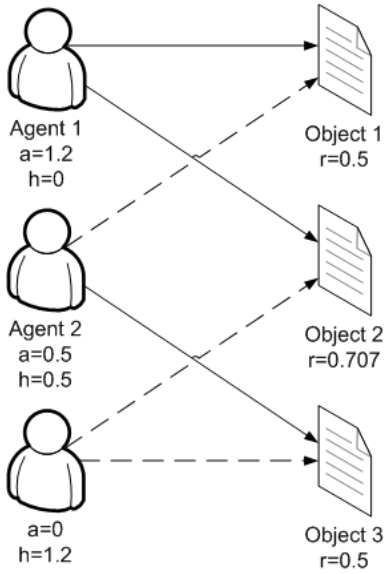

Figure 4.3: How agents provision (solid line) and evaluate (dashed line) objects. All score are computed: authority a, hub h and reputation r. Constant α set to 0.5.

The EigenRumor algorithms uses two vectors known from the HITS 4.2.2, but we need an another one to better description agent-to-object associations. [FIS05]

Authority score - agent property The authority score indicates to what level agent i provided objects in the past that followed the community directions. It is considered that the higher the score, the better the ability of the agent to provide objects to the community. We define the authority vector $\vec{a}$ as a vector containing all authority score a_i for agent i, $i = 1, \ldots, m$.

Hub score - agent property The hub score indicates to what level agent i submitted evaluations that followed the community directions on other past object. It is considered that the higher the score, the better the ability of the agent to contribute evaluations to the community. We define the hub vector $\vec{h}$ as a vector that contains the hub score h_i for agent i, $i = 1, \ldots, m$.

Reputation score - object property Thereputation score indicates the level of support from agents that follow community directions. It is considered that the higher the score means that the object better conforms to the community direction. We define the reputation vector $\vec{r}$ as a vector that contains the reputation score r_j for object j, $j = 1, \ldots, n$.

All these vector influence each other. We have four natural assumption about it.

Assumption 1: The objects that are provided by a good authority will follow the direction of the community.

Assumption 2: The objects that are supported by a good hub will follow the direction of the community.

Assumption 3: The agents that provide object that follow the community direction are good authorities of the community.

Assumption 4: The agents that evaluate object that follow the community direction are good hubs of the community.

These assumptions are similar to common assumptions in our life. If someone likes his favourite author, and he considers his books interesting, there is a high probability, that he will buy a new book from this author. And even without any reference to it's quality. That man naturally assumes that each new book written by his favourite author will be also worth to be read. And therefore, we think that it is suitable to use the analogy in ranking algorithm for blogs. The new article written by a good author will get the higher reputation rank than a new posting from lower ranked author. We can transfor above assumptions to corresponding equations.

$$\vec{r} = P^{\mathsf{T}}\vec{a} \tag{4.2}$$

$$\vec{r} = E^{\mathsf{T}}\vec{h} \tag{4.3}$$

$$\vec{a} = P\vec{r} \tag{4.4}$$

$$\vec{h} = E\vec{r} \tag{4.5}$$

We have two equations 4.2 and 4.3 which have $\vec{r}$ on the left side. We can merge them by using a convex combination with a constat $\alpha \in [0, 1]$:

$$\vec{r} = \alpha P^{\mathsf{T}}\vec{a} + (1 - \alpha)E^{\mathsf{T}}\vec{h} \tag{4.6}$$

With the constat α we can adjust the weight of authority and hub score. We have now three equations, (4.4), (4.5) and (4.6) that recursively define three score vectors. We will expand $\vec{a}$ and $\vec{b}$ in (4.6) with appropriate right sides from equations (4.4) and (4.5).

$$\vec{r} = \alpha P^{\mathsf{T}} P\vec{r} + (1-\alpha)E^{\mathsf{T}} E\vec{r} = (\alpha P^{\mathsf{T}} P + (1-\alpha)E^{\mathsf{T}} E)\vec{r} \tag{4.7}$$

Let $S = (\alpha P^{\mathsf{T}}.P + (1-\alpha)E^{\mathsf{T}}E)$. Then S is a stochastic square matrix, because sum of all elements in every column is 1, and therefore when we will apply iteration by (4.7), $\vec{r}$ will converge to the principal eigenvector of S. We will simultaneously compute all others score vectors.

Listing 4.1 EigenRumor algorithm scheme

$\vec{a}_0 \leftarrow (1, \ldots, 1)$
$\vec{h}_0 \leftarrow (1, \ldots, 1)$
while $\|\vec{r}_i - \vec{r}_{i-1}\|_2 < \epsilon$ for some small ϵ **do**
  $\vec{r}_i \leftarrow \alpha P^{\mathsf{T}} a_i^{\mathsf{T}} + (1-\alpha)E^{\mathsf{T}} h_i^{\mathsf{T}}$
  $\vec{r}_{i+1} = \frac{\vec{r}_i}{\|\vec{r}_i\|_2}$
  $\vec{a}_{i+1} = (P\vec{r}_{i+1})^{\mathsf{T}}$
  $\vec{h}_{i+1} = (E\vec{r}_{i+1})^{\mathsf{T}}$
end while

4.2.4 EigenRumor adjustments

We are convinced, that readers' interests are better reflected in their votes and comments than in links they are creating. People do not link every page or article they like. Most usuall expression of people's favour to a post is their positive vote. Leaving a comment is also useful indication of some kind of interest. So, we replace standard method of evaluation in EigenRumor from linking to voting. We also include comments as a part of the evaluation.

We define the evaluation matrix as a sum of two matrices: a *vote matrix* V and a *comment matrix* C.

$$E = \beta V + (1-\beta)C \tag{4.8}$$

The constant β again determines the balance between votes and comments. If $\beta = 0$ only votes are taken into account and vice versa. Values of matrices V and C are defined similarly as matrix E. If a blogger i gave a positive vote for a post j then $V_{ij} = 1$. Value C_{ij} is expressed more complicated. We require $C_{ij} \in \langle 0, 1\rangle$ and we need to include there all comments by the blogger i. We also want to consider time factor in the comment evaluation to prioritise posts with vigorous discussions.

We resolve the second condition with formula (4.9) where $Comm(i, j)$ is a set of all comments posted by the agent i in the object j. $\Delta T(l)$ in the exponent means time difference between current time and time of comment creation in days. We need a constant γ which determines the speed of falling

of C, $\gamma = 10$ seemed to be satisfactory value.

$$C'_{ij} = \sum_{l \in Comm(i,j)} e^{-\frac{\Delta T(l)}{\gamma}} \tag{4.9}$$

Now, we need to fulfil the first condition. C_{ij} has to be in $\langle 0, 1 \rangle$, because we want to preserve the properties of the generic EigenRumor and members of the evaluation matrix E should be in range from 0 to 1. For this purpose we used a function $f(x) = 1 - \frac{1}{x+1}$. This function converge to 1 in the infinity. The final formula for C_{ij} is:

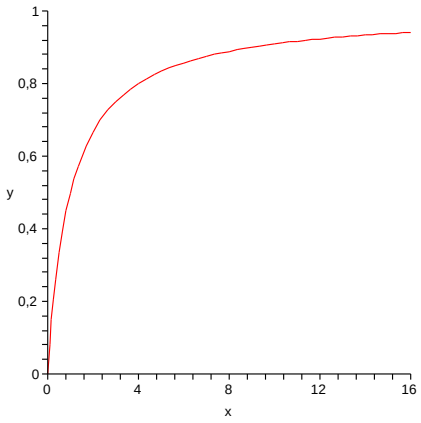

Figure 4.4: Graph of function $f(x) = 1 - \frac{1}{x+1}$

$$C_{ij} = f(C'ij) = 1 - \frac{1}{\sum_{l \in Comm(i,j)} e^{-\frac{\Delta T(l)}{\gamma}} + 1} \tag{4.10}$$

Now the formula (4.6) can be slightly modified into:

$$\vec{r} = \alpha P^{\mathrm{T}} \vec{a} + (1 - \alpha)(\beta V + (1 - \beta)C)^{\mathrm{T}} \vec{h} \tag{4.11}$$

Another improvement, we have done, is enabled negative evaluations. We run two instance of the EigenRumor. One computes positive scores and the another computes negative scores. The final reputation score is difference between positive and negative scores. The authority score is similar. The hub score is important only for the computation, so we need not to express it as a one value.

Another extension that we have implemented is support for votes of anonymous users. We creat a one virtual user assigned to all non-logged guests. This user has no authority score but he has non-zero hub score which is not computed by EigenRumor but it is manually set to a constant value or to a half of median of all hub scores.

4.2.5 Personalisation

There are some people with closer opinions in every community. They consider interesting and useful other articles or authors than general community opinion. Traditional ranking algorithms, as we have described, do not offer them best articles that could be proposed. Therefore, we think that personalised ranking algorithms will be in future the main stream of ranking algorithms and so we have designed some improvements towards the personalisation in our portal.

We used information collected by generic EigenRumor algorithm. Matrices E and P provide useful data for expression of similarities in opinions in the community.

Neighbourhood

The first individual relation, we are computing, is *neighbourhood of bloggers*. Every blogger (agent) has a set of neighbours (agents) which have similar opinions expressed by their votes. For this purpose, we used information stored in *evaluation matrix* E.

Two agents, $agent_i$ and $agent_j$ are neighbours if they have similar evaluation vectors $\vec{e_i}$ and $\vec{e_j}$. We will need a neighbourhood function ν for defining neighbourhood relation $\mathcal{N}$. The $agent_i$ and $agent_j$ are in neighbourhood relation $\mathcal{N}$ if $\nu(\vec{e_i}, \vec{e_j}) > k$ where k is some threshold.

We assume a system with 3 agents and 4 objects. Provisioning matrix is not important now but evaluation matrix is:

$$\mathbf{E} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ -1 & -1 & 0 & 1 \end{pmatrix}$$

This matrix includes also negative evaluations what is not possible in original algorithm but we have enabled it in extension described in previous subsection 4.2.4. We see that $\vec{e_1}$ and $\vec{e_2}$ are quite similar but $\vec{e_3}$ is entirely different. We define function of neighbourhood $\nu(\vec{e_i}, \vec{e_j}) = \frac{\vec{e_i}.\vec{e_j}}{||\vec{e_i}||.||\vec{e_j}||}$. Where numerator is a scalar multiplication of both evaluation vectors, which computes the number of equal votes for posts, where the both bloggers voted. The denominator normalises value of nominator with both lengts of vectors .

It is obvious, that ν is a symmetric function; $\nu(\vec{e_i}, \vec{e_j}) = \nu(\vec{e_j}, \vec{e_i})$. In fact, ν is nothing other than a cosines of angle between $\vec{e_i}$ and $\vec{e_j}$. And indeed, cosines has necessary attributes we need from neighbourhood function. If two vectors are identical i.e. the angle between them is 0 and $cos(0) = 1$ and vice versa if there are entirely different vectors, $\vec{e_j} = -\vec{e_i}$, so $cos(\pi) = -1$.

Matrix $\mathbf{N}$ for evaluation matrix $\mathbf{E}$, where $\mathbf{N}_{ij} = \nu(e_i, e_j)$:

$$\mathbf{N} = \begin{pmatrix} 1 & 0.81 & -0.81 \\ 0.81 & 1 & -0.66 \\ -0.81 & -0.66 & 1 \end{pmatrix}$$

If we have computed a set of neighbours for each user, it would be useful to propose him the posts, which have votes from his neighbours but he did not vote for them. If two users have very similar evaluation vectors, it is probable, they have the same attitude towards a new post.

The similar conception is applicable also for *posts neighbourhood*. If a reader has read a post and he liked it, it would be useful to propose him some similar posts. We consider similar posts, that posts, which have a vote from a very similar group of users. The nominators in the previous case were members of a matrix $\mathbf{M}$ where $\mathbf{M} = \mathbf{E}.\mathbf{E}^{\mathsf{T}}$. This matrix $\mathbf{M}$ is a square matrix with dimension $n \times n$, where n is the number of bloggers. In the neighbourhood of posts, we will need a square matrix with dimension $m \times m$, where m is the number of all posts. A matrix $\mathbf{K} = \mathbf{E}^{\mathsf{T}}.\mathbf{E}$ satisfy the requirements. The denominators are again products of multiplication lengths of selected vectors.

When we have computed sets of post neighbours, we propose to reader the similar posts, which he has not read.

4.2.6 BlogRank

BlogRank algorithm was introduced by a Greek team in [KSV06]. BlogRank is a generalised version of PageRank. The output of this algorithm is a ranking of all weblogs in a dataset. Authors tried to make the graph of blogosphere more dense. Apart of explicit hyperlinks among nodes, they enhance the graph with implicit edges based on similarity in topics and contributors or difference in time of creation. The connected graph of weblogs is expanded by adding bidirectional edge between each blogs sharing same categories, user and external resources. Each edge has assigned a weight which was tuned by experiments.

They did tests of BlogRank on a sample weblog dataset provided by Nielsen BuzzMetrics, Inc. and they implemented an experimental search engine for the Greek portion of blogosphere. According to their article, BlogRank had the

best result in Success Index[1] presented in [KS03]. PageRank acquired 0.158, XRank (probably. they tought iRank) 0.353 and BlogRank 0.553.

The BlogRank of a weblog A is given by the following formula:

$$B(A) = (1-d) + d(FN(U_{1\to A})B(U_1) + \ldots + FN(U_{n\to A})B(U_n)) \tag{4.12}$$

An obvious inspiration from equation (4.1) is present here. $B(A)$ is a the BlogRank of blog A. The constant d is the damping factor, nNormally set to 0.85. $FN(U_{i\to A})$ expresses a possibility of transition from weblog i to weblog A, i.e. how i fancies A. For $FN(U_{i\to j})$ the following equation holds:

$$\sum_{j=1}^{t} FN(U_{i\to j}) = 1 \tag{4.13}$$

where t is number of outgoing links from the node i. If $FN(U_{i\to j}) = \frac{1}{t}$, we get exactly the formula (4.1). In BlogRank, it is given following formula for FN:

$$FN(U_{i\to j}) = \frac{F_{i\to j}}{\sum_k (F_{i\to k})} \tag{4.14}$$

and where

$$F_{i\to j} = L_{i\to j} + w_T T_{i\to j} + w_U T_{i\to j} + w_N N_{i\to j} + w_D D_{i\to j}$$

and L is the number of links from the blog i to the blog j, T is the number of common tags or categories and U is the number of users who have written in the both blogs. Number N is count of common links to external pages and D equals to $\frac{20*60}{\delta}$ where δ is average of posting time difference in minutes. Values w_T, w_U, w_N, w_D are weights assigned to corresponding T, U, N, D. Weights was set up experimentally as:

$$\begin{aligned} w_T &= 1.7 \\ w_U &= 1.1 \\ w_N &= 4.8 \\ w_D &= 0.4 \end{aligned}$$

Unfortunately, search engine they have developed was not available at that time when we was writing this work and we could not test their work more rigorously.

4.2.7 B2Rank

Another ranking algorithm based on the same principles as PageRank is B2Rank. This rank was designed and tested at Amirkabir University of Technology in Tehran [THM07]. Authors take into account that blogs updated more regularly

[1] A measure of users' satisfaction with search results

tend to be more popular [CK06] and also an amount of comments is an suitable indication of the community interest [MG06]. They consider in B2Rank behavioural features of users such as blog updating rate, comment putting and different types of citations and their creation times.

B2Rank assigns two scores for each weblog: personality score and operation score. Scores are divided because of it is considered two types of links: blogrolls link and citation links. Citation links are object-to-object links and blogrolls are agent-to-agent links in the terminilogy we were using in section 4.2.3.

There are used two types of graphs for B2Rank computation. *Blogroll graph* $\mathcal{B}$ consists of nodes representing blogs and edges which correspond to blogrolls links. Second graph is called *entries graph.* Nodes in entries graph $\mathcal{E}$ are blog posts and edges are explicit and implicit links. The explicit link is a citation and the implicit link is created between user who publish a comment and commented post. Authors do not present in their work which node in graph $\mathcal{E}$ is the source of this link because there are only blog entries in the set of nodes. The time function $T(i, j)$ expresses the time of link creation for each edge in the graph.

Blogroll's link has assigned two weights corresponding to two factors, called activity and attention. Activity conveys the frequency of blog updating. Activity score function for blog i is computed as follows:

$$A(i) = \frac{count\ of\ all\ posts\ in\ blog\ i}{day\ since\ creation\ of\ blog\ i}$$

Blog attention is related with the number of received comments. Blogs with more comments have a better attention score. Attention score function is defined with formula:

$$N(i) = \frac{Com(i)}{count\ of\ all\ posts\ in\ blog\ i}$$

where $Com(i)$ stand for the count of all comments in the blog i.

The main formula of B2Rank for a blog x is:

$$B2Rank(x) = PersonalityScore(x).OperationScore(X) \tag{4.15}$$

PersonalityScore is computed by well known scheme derived from PageRank.

$$PersonalityScore(x) = d\left(\sum_{(y,x)\in\mathcal{B}} PersonalityScore(y).NB(y,x)\right) + (1-d) \tag{4.16}$$

Where d is the same damping factor as in (4.2.1), y is a blog which links to x in its blogroll and $NB(y, x)$ expresses the possibility of transition from y to x.

This possibility is computed by following formula:

$$NB(y,x) = \frac{B(y,x)}{\sum\limits_{(y,x)\in\mathcal{B}} B(y,x)} \tag{4.17}$$

$B(y,x)$ is the weight of blogroll link from y to x.

$$B(y,x) = W_a A(x) + W_n N(x)$$

Where W_a and W_b are weights assigned to the activity score and the attention score respectively.

OperationScore(x) denotes an average quality of blog posts in a certain blog x. We need to compute post score $EB(e)$ for each post e in the blogosphere. EB is computed similarly as *PersonalityScore*.

$$EB(e) = d\left(\sum_{(b,e)\in\mathcal{E}} EB(b).NE(b,e)\right) + (1-d) \tag{4.18}$$

$EB(b)$ means the post score of the entry b which links to the entry e. $NE(b,e)$ expresses similarly to NB the possibility of reader's movement from the entry b to the e. This possibility is computed by formula

$$NE(b,e) = \frac{E(b,e)}{\sum\limits_{(b,e)\in\mathcal{E}} E(b,e)}$$

where $E(b,e)$ is computed weight of link between b and e. This weight is assigned in the following way:

$$E(b,e) = W_c.Com(e) + W_t.e^{-\Delta T(b,e)}$$

W_c and W_t are weights for $Com(e)$ and $\Delta T(b,e)$. $\Delta T(b,e)$ is time delay of citation in post b to post e, compared with other citations to post b. Now we are able to compute the operation score of blog x.

$$OperationScore(x) = \frac{\sum\limits_{\forall e\in x} EB(e)}{\sum\limits_{\forall e\in x} 1} \tag{4.19}$$

Results

Authors implemented this algorithm on data collected by BlogScience Research Group. They performed PageRank and B2Rank algorithms on 22,000

Persian blogs with 378,700 posts and 1,275,561 collected in 15 months. According to reports of a group of users who did independent tests and who did not know which rank system gave them result, 38% of them did not notice any significant difference, 51% evaluated B2Rank's results as better and 11% worse.

Chapter 5

Implementation

Implementation of the whole project was strongly influenced by technologies we wanted to try. We had to design a specific data model which would be the best for the application with data stored in the XML native database. We started with 3 main decisions; a programming language, a database and a data processing/presentation method.

Our first decision was to build up our blog portal on eXist (see. 2.3.2). It was in November 2006, and eXist 1.1 seemed to be the best choice. Then, we decided to use PHP as a main programming language. The PHP is a powerful scripting language suitable for fast application development. It is maybe not the fastest language but flexibility and rapid development were more important requirements in our experimental project than the performance. However, eXist has great Java API, we also founded a solution for PHP connection to the database. We used the SOAP protocol for the communication with the database.

We also needed a tool for manipulation with XML retrieved from the database. We consider XSLT to be the optimal solution, because we can separate the presentation layer easily from the logical layer as in contemporary programming paradigm MVC. Of course, the XSLT is primary dedicated for such purpose.

We expected that some complications can arise, especially with so unproved technology as the NXD. We thought of it and therefore we tried to make an universal interface (fig. 5.1) for NXDs. We wanted to have a database independent application also for experiments with various database engines. As we have described in section 2.3.2, it was far-sighted step.

We wrote a class Exist for work with eXist which implemented NXDapi interface. It was very simple to write a Sedna class which implements the same interface in February 2008. And although the update languages in eXist and Sedna are not completely compatible, the transition from eXist to Sedna was almost painless.

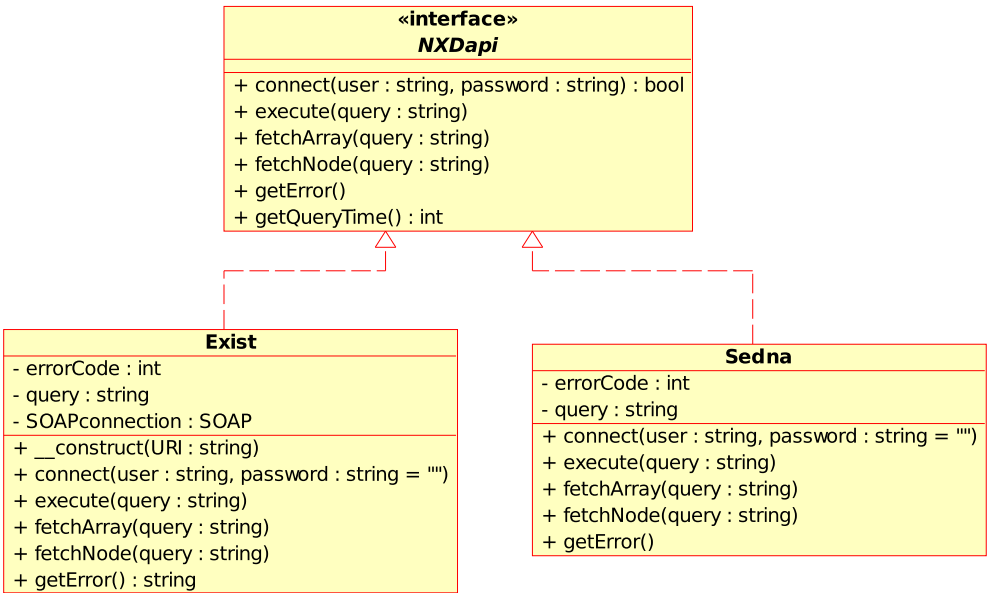

Figure 5.1: Interface for XML databases

5.1 Data model

All blogging applications have to store similar data. It is necessary to have stored info about users, their postings and also comments. Comments are naturally tree-like structured data and also other data can be inserted into a tree. We have created a forest of users trees. This forest is a one collection (see 2.2.1) called "weblog". We assigned a new XML document for each user. The `user` element is a root element of these documents and all data belonging to user are its descendants.

We have divided user's data into 3 main elements. All information about user's account are joined in element `account` and its subelements. User's private information such as real name, "about" information and also the user's ranks are situated in the `info` element. And the third main group are data under element `blog`. There can be found all users posts with comments with other necessary records. Each posting or commnet has an unique id (identifier) in the collection. We are able therefore address place where to attach a comment only with the one identifier. It facilitate a creation of a custom blog by students of 1-AIN-636 Modern Approaches to Web Design. We implemented a special interface where they can create own blog only by using XSLT, HTML and CSS. We provided them only a function for updateting their XML document. Their blogs with their own design was a part of evaluation in this course. These students helped us to test our application. They produced a high load on the server and we were able to identify problematic parts faster.

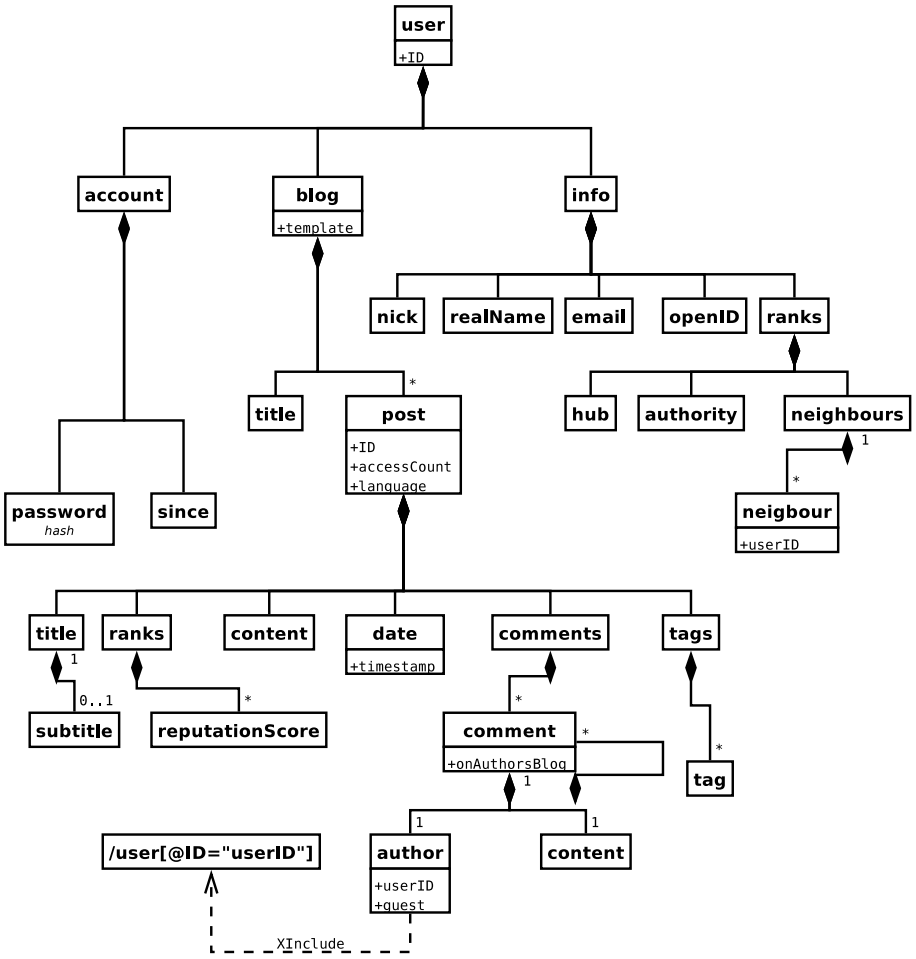

Figure 5.2: Data model scheme

Partially independent part of our portal is a ranking subsystem. It has own storage engine because of two reasons. First reason is relational character of stored data. Second reason is that in the time of ranking implementation we were been using eXist which produced many errors by frequent updates and voting with computation of ranks makes a lot of requests on the server. We used a MySQL server for storing data necessary in EigenRumor algorithm. Provision and evaluation matrices have own simple tables and they can be easily and quickly queried for a record. MySQL server is faster (approx. from 10 to 100 times) and more reliable for this purpose.

5.2 Main application

We planned to develop a fully featured community blog portal, that would provide common blogging tools, comparable with contemporary blog portals. We wanted to create a custom space where everybody from FMFI (also foreign users are welcome) can publish news, opinions and suggestions. FMFI does not have any community platform where students and teachers can communicate without any restrictions and at the same level. The portal `blog.matfyz.sk` is now a perspective project with a growing base of users and there are about 250 unique visitors and 500 pageviews each day.

For a successful implementation we created as UML model which helped us to divide our work into separate blocks. We attach the simplified use-case diagram in the appendix, that shows main provided functions. This diagram does not include a part responsible student[1] interface.

We designed two classes for a blog processing. The class BlogReader serves to reading and selecting data from a certain blog. Second class, BlogWriter, manipulates with XML in the database and the blog data should be modified only through this class. Another class User is used for both operations, for setting and getting attributes of the actor user. PortalReader is class similar to BlogReader, it serves to retrieving data from the whole portal.

Whole application is managed by Controller, a class which delegates suitable object to do their tasks. Class Page is responsible for the presentation. It chooses an appropriate source of the data and requests the XSLT class for an output XSL template. Descendants of DataSource return the XML data retrieved from the database. General functions necessary in the whole system are provided by the class System.

RankManager is a class dedicated for operations that belongs to EigenRumor algorithm. It does not compute the ranks, because we implemented the core of the EigenRumor algorithm in C++. We tried to implement it in PHP, but PHP appeared to be insufficiently slow language for operations on the large matrices. The C++ accelerated the ranking algorithm about 60 time against PHP. The computation of ranks are executed every half an hour by a cron daemon. In the time, when we were using eXist, we could not execute computation too often because the ranking algorithm regularly corrupted indices.

EigenRumor algorithm can produce as bad results as any trivial algorithm based on voting. Important part of this algorithm is right choice of the constants. Constant α (in listing 4.1 is α replaced with k) determines how much to consider the bloggers's authority in the finall reputation score for a post. If $\alpha = 0$ no authority score is used and reputation score depends only on votes.

[1] students of Modern Approaches to Web Design

Computed ranks are used for top 10 list and also as search an order criterium.

For $\epsilon = 10^{-6}$ in algorithm's listing 4.1, vector r converges in about 20 iterations. This computation must be redone after each addition of a new object in the system. We can save some iterations by initialising starting vectors to values from the last execution of algorithm. Because a small change in P and E will influence the result very slightly. By this step, we reduced number of the iterations to 10-30%. In many cases the minimal count of the iterations is necessary. (2-4 iterations). We have observed that speed of the convergation on our collected data is connected to choice of α. With higher α is needed more iterations.

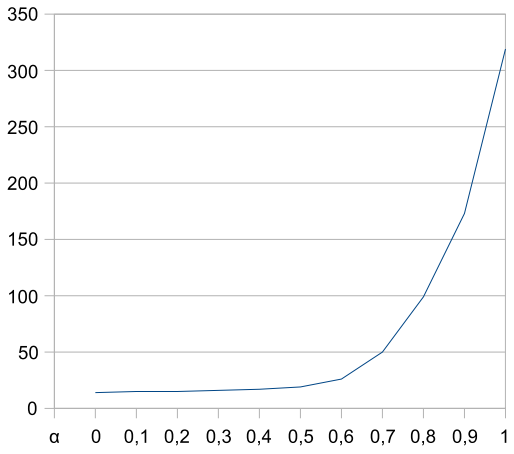

Figure 5.3: Dependency the iterations count on α

We choosed the DokuWiki syntax (`http://wiki.splitbrain.org/wiki:syntax`) was for writing posts and comments on our portal. We implemented our DokuWiki parser (`http://stettner.blog.matfyz.sk/p1-syntax-i`) with support of source code highlighting. We also added a support for LaTeXoutput, which might be useful especially for people from FMFI community.

Listing 5.1 Our implementation of EigenRumor Algorithm in C++

```
while ((r1_old-r1).L2norm()>0.000001)
{
  r1_old = r1;
  r1 = p.transpose()*k*a1.transpose()
       + e1.transpose()*(1-k)*h1.transpose();
  r1 = r1*(1/r1.L2norm());
  a1 = (p*r1).transpose_matrix();
  h1 = (e1*r1).transpose_matrix();
  /* setting the anonymous user hub score */
  h1.value(0,rows-1) = GUEST_HUBSCORE;
  r1 = r1.transpose_matrix();
}
```

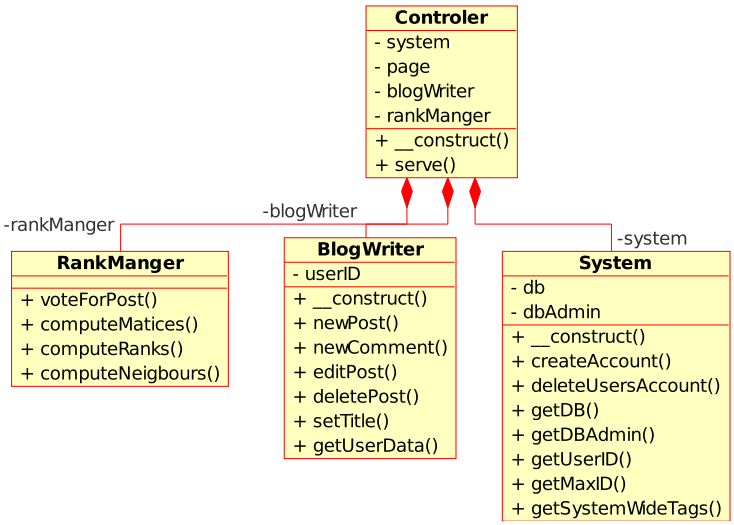

Figure 5.4: Class diagram for data manipulation

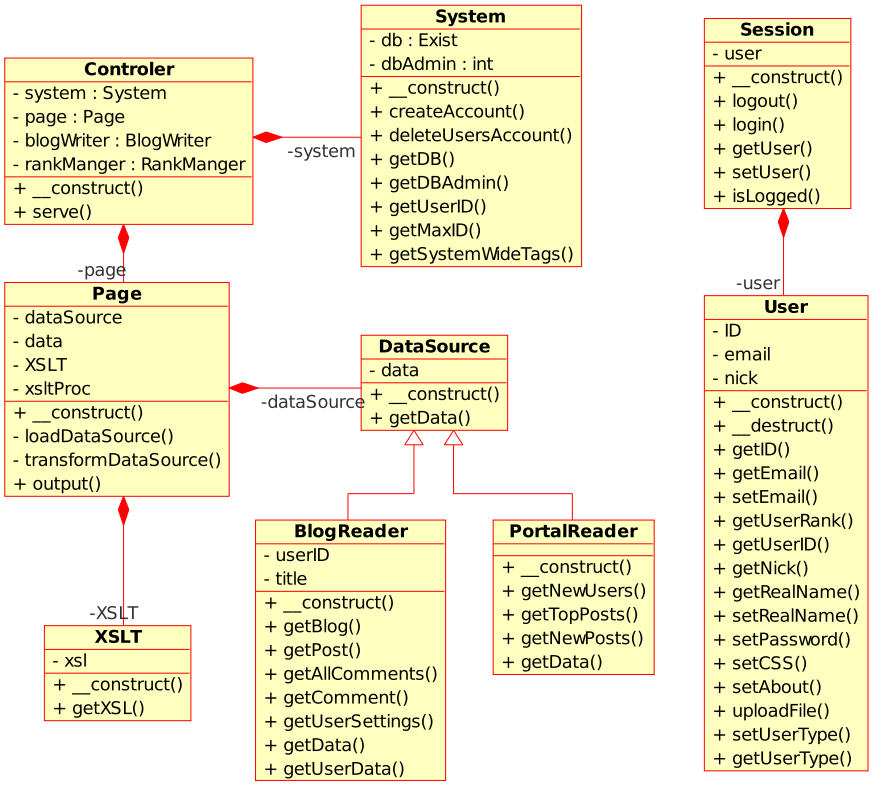

Figure 5.5: Class diagram for the presentation layer

Figure 5.6: blog.matfyz.sk title page

Chapter 6

Conclusion

The product of our work is a fully functional community portal launched in September 2007. It supports all common blogging features and it has a potential of being the popular and useful communication platform for students and teachers from FMFI, but also for general public. There are 250 unique visitors each day and we now have circa 140 registered users with 600 posts published.

We described the current status of native XML databases in our work with three selected products. Two of them were used and tried as a storage system in our project, a community blog portal `blog.matfyz.sk`. Before we started with this work, we were able to induce several hypotheses on avdantages and disadvantages of NXDs from the literature. Almost all of them came true, even though it would be better if only advantages were confirmed. The drawbacks with updating and indexing brought us a lot of additional work we had to deal with.

We assumed that a tree structured data model reflects the reality more closer and more naturally than relational model. Data are not decomposed and they stay together. Therefore, they can be effectively used. These advantages were proven in our work and we have succeeded to design a simple, easily readable and self-describing data model. The XML data model stored in a schema-less collection appeared to be flexible enough. This was confirmed by additional implementation of special interface for students which were then able to create their own blog layout with XSLT technology. Thus this work has also served in the educational process.

Our next assumption was that processing of tree structured data is considerably more effective by usage of XML databases. It is necessary to walk through records several times (in common case as much as the depth of the tree is) to retrieve a tree structured data from relational database. In a XML database one query will be enough. All data for displaying a one blog can be retrieved by only a one simple query on an XML database. It is true that

queries on an NXD take more time, but there is a significant saving in the amount of necessary queries. It has a result that total time for page generation is very similar to applications powered by RDBMS and the last user probably will not notice any performance difference.

We are convinced that XML databases are useful tools which can be used in many application types. But NXDs are still in their beginnings and in some cases there is still need for intensive development. The Sedna project seemed to be the best solution for us. The third release of Sedna is quite stable and moreover our reports have contributed to improvements in this release. We also tried eXist which has rich features but there are still some problems with the indexing engine. We appreciate that eXist offers two additional operators for full-text search ("&=" and "|=") which we think would be useful to include in XQuery specification.

The usability of XML databases will certainly improve if any form of update language become standardised. Untill that moment, every database will use their own language not necessarily compatible with others.

In the Chapter 4, we elaborated an overview of blog specific ranking algorithms. We studied the EigenRumor algorithm closer and we proposed adjustments and changes which improved this algorithm's results in our conditions. Only this algorithm, from the group of examined algorithms, assigns higher ranks for new posts written by popular blogger which has published already appreciated posts. Our modified EigenRumor works on much more denser graph of objects than the original one because we replaced the evaluation method from linking to voting and we also take into account the amount and the vigorousness of discussions.

We also suggested a method for proposing interesting articles related to the article the user is currently reading. We are curious about how the solutions proposed in this work will perform with further growing users' base and the number blog posts in the portal.

6.1 Future work

A blog portal is an excellent place for research projects. We dealt only with a few aspects which our blog portal had proposed and there are still many issues which should be solved. Though, we implemented a spam protection reCAPTCHA but it did not work always reliable. An interesting topic for the future would be a collaborative filtering algorithm for comments. Another worthy work is to exploit ontologies and semantics technology in order to categorisation of posts.

Appendix A

Related graphs

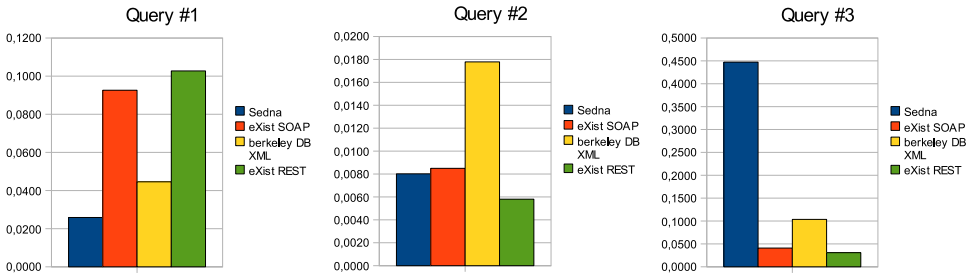

Figure A.1: Queries performance 1 – 3

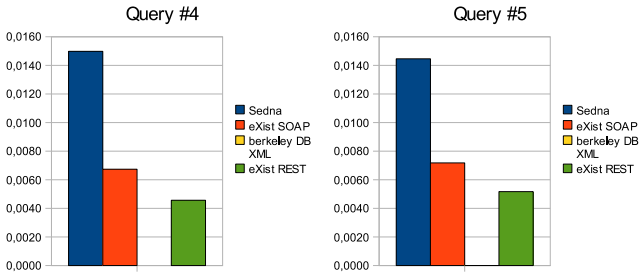

Figure A.2: Queries performance 4 – 5

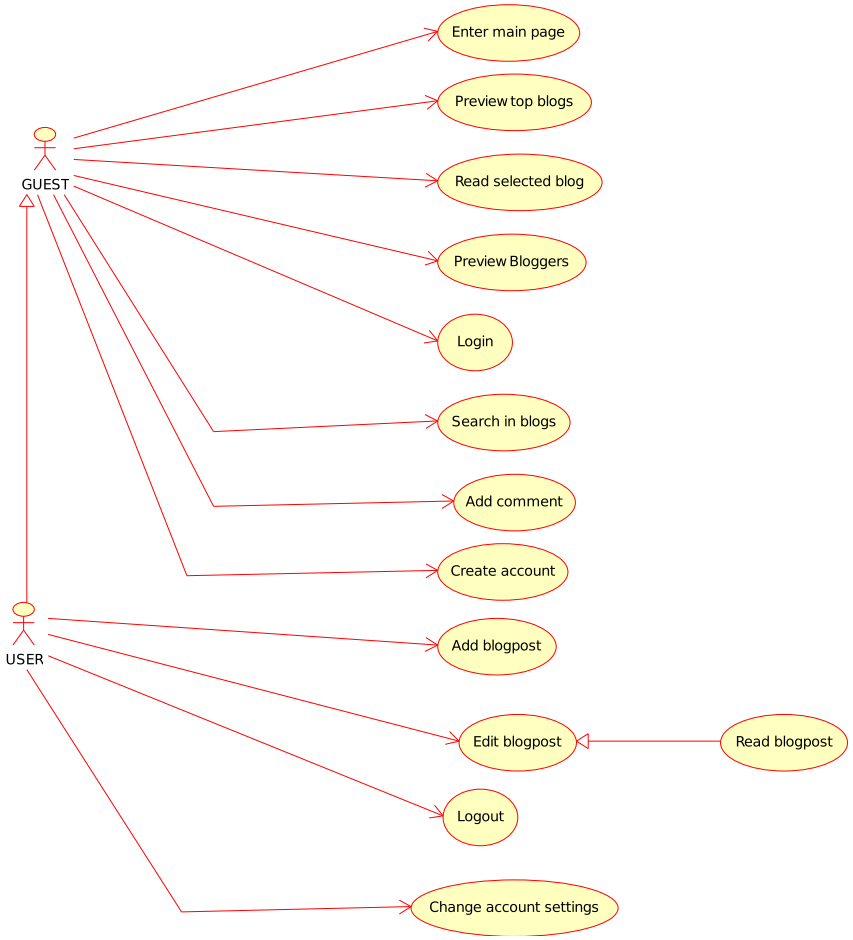

Figure A.3: Simplified use-case diagram for blog.matfyz.sk

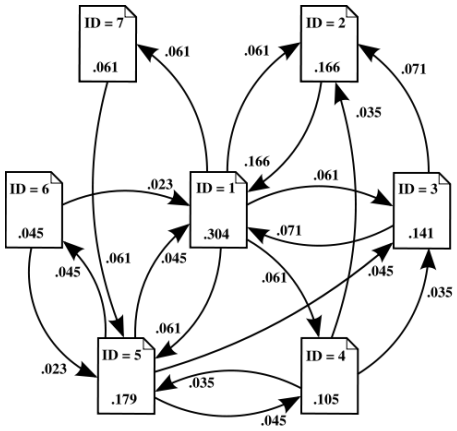

Figure A.4: Simple schema of PageRank computation

Glossary

API	Application Programming Interface
Blog	A weblog (or blog) is a frequently updated web page containing items of information arranged in reverse chronological order. A weblog can typically take the form of a diary, journal, what's new page, or links to other web sites.
Blogger	Person who writes blogs
Blogging	Activity of mantainig a blog
Cron	Cron is a command scheduler that resides on your server and executes commands at specified time intervals.
CSS	Cascading Style Sheets
DOM	The Document Object Model (DOM) is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.
FMFI	Faculty of Mathematics, Physics and Informatics
MVC	Model-View-Controller
NXD	Native XML database
PCDATA	PCDATA is a token used in an element declaration to declare the element as having mixed content (character data, or character data mixed with other elements)
RDBMS	A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd. Most popular commercial and open source databases currently in use are based on the relational model.
RSS	Really Simple Syndication
SAX	Simple API for XML.
SEO	Search engine optimizing
SOAP	SOAP is a protocol for exchanging XML-based messages over computer networks, normally using HTTP/HTTPS
W3C	The World Wide Web Consortium (W3c) is the main international standards organization for the World Wide Web

Wiki	A wiki is a collection of web pages designed to enable anyone who accesses it to contribute or modify content, using a simplified markup language. Wikis are often used to create collaborative websites and to power community websites
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Bibliography

[ACHM05] Paolo Avesani, Marco Cova, Conor Hayes, and Paolo Massa. Learning contextualized weblog topics. In *Proceedings of the 2nd WWW workshop on the Weblogging Ecosystem*, 2005.

[AZAL04] Eytan Adar, Li Zhang, Lada A Adamic, and Rajan M. Lukose. Implicit structure and the dynamics of blogspace. *Workshop on the Weblogging Ecosystem, 13th International World Wide Web Conference*, 2004.

[BB06] DL Byron and Steve Broback. *Publish and Prosper: Blogging for Your Business*. New Riders, June 2006.

[BCFF03] Scott Boag, Don Chamberlin, Mary F. Fernández, and Daniela Florescu. Xquery 1.0: An xml query language. Technical report, W3C, 2003. W3C Working Draft 12 November 2003, available online, at `http://www.w3.org/TR/2003/WD-xquery-20031112/`.

[BH98] K. Bharat and M. Henzinger. Improved algorithms for topic distillation in hyperlinked environments, 1998.

[BN06] Bettiba Berendt and Roberto Navigli. Finding your way through blogspace: Using semantics for cross-domain blog analysis. In *Proceedings of the AAAI 2006 Symposium on Computational Approaches to Analysing Weblogs.*, page 8, 2006.

[Bou05] Ronald Bourret. Xml and databases. Available online, at `http://www.rpbourret.com/xml/XMLAndDatabases.htm`, September 2005.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *WWW7: Proceedings of the seventh international conference on World Wide Web 7*, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[CDG07] Carlos Castillo, Debora Donato, and Aristides Gionis. Estimating number of citations using author reputation. In Nivio Ziviani and Ricardo A. Baeza-Yates, editors, *SPIRE*, volume 4726 of *Lecture Notes in Computer Science*, pages 107–117. Springer, 2007.

[Cha02] Nicholas Chase. *XML Primer Plus*. Sams, December 2002.

[CK06] Edith Cohen and Balachander Krishnamurthy. A short walk in the blogistan. *Computer Networks*, 50(5):615–630, 2006.

[CR07] Henning Christianses and Maria Rekouts. Integrity checking and maintenance with active rules in xml databases. *24th British National Conference on Databases : proceedings*, pages 59–67, 2007.

[Cuo06] Nguyen Viet Cuong. Xml native database systems review of sedna, ozone, neocorexms. Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2006.

[FGK06] Andrey Fomichev, Maxim Grinev, and Sergei D. Kuznetsov. Sedna: A native xml dbms. In Jirí Wiedermann, Gerard Tel, Jaroslav Pokorný, Mária Bieliková, and Julius Stuller, editors, *SOFSEM*, volume 3831 of *Lecture Notes in Computer Science*, pages 272–281. Springer, 2006. Available online, at `http://www.ispras.ru/~grinev/mypapers/sedna.pdf`.

[FIS05] Ko Fujimura, Takafumi Inoue, and Masayuki Sugisaki. The eigenrumor algorithm for ranking blogs. In *Procs. of 2nd Annual Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics*, page 6, 2005.

[Hal05] David Hall. An xml-based database of molecular pathways. Master's thesis, Linköpings universitet, Department of Computer and Information Science, 2005.

[Har05] E. R. Harold. Managing xml data: Native xml databases. Available online, at: `http://www-128.ibm.com/developerworks/xml/library/x-mxd4.html?ca=dnt-623`, Jun 2005.

[Hil05] Jefrey Hill. The voice of the blog: The attitudes and experiences of small business bloggers using blogs as a marketing and communications tool. Master's thesis, University of Liverpool, 2005.

[HKP+05] Susan C. Herring, Inna Kouper, John C. Paolillo, Lois Ann Scheidt, Michael Tyworth, Peter Welsch, Elijah Wright, and Ning Yu. Conversations in the blogosphere: An analysis "from the bottom up".

In *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 4*, page 107.2, Washington, DC, USA, 2005. IEEE Computer Society.

[KC06] Peter Kuhns and Adrienne Crew. *Blogosphere: Best of Blogs.* Que, January 2006.

[Kle99] Jon Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM 46*, 1999.

[KS03] Apostolos Kritikopoulos and Martha Sideri. The compass filter: Search engine result personalization using web communities. In Bamshad Mobasher and Sarabjot S. Anand, editors, *ITWP*, volume 3169 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2003.

[KSV06] Apostolos Kritikopoulos, Martha Sideri, and Iraklis Varlamis. Blogrank: ranking weblogs based on connectivity and similarity features. In *AAA-IDEA '06: Proceedings of the 2nd international workshop on Advanced architectures and algorithms for internet delivery and applications*, page 8, New York, NY, USA, 2006. ACM.

[Leh01] Patrick Lehti. Design and implementation of a datat manipulation processor for an xml query language. Master's thesis, Technische Universität Darmstadt, 2001.

[LM04] A.N. Langville and C.D. Meyer. Deeper inside pagerank. *Internet Mathematics*, 1(3):335–380, 2004.

[Mei02] Wolfgang Meier. exist: An open source native xml database. Available online, at `http://www.old.netobjectdays.org/pdf/02/papers/ws-webdb/01-Meier.pdf`, 2002.

[Mey00] Carl D. Meyer. *Matrix analysis and applied linear algebra.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[MG06] Gilad Mishne and Natalie Glance. Leave a reply: An analysis of weblog comments. 3rd Annual Workshop on the Weblogging Ecosystem, 2006.

[MT07] MODIS-Team. Sedna programmer's guide. Available online, at `http://modis.ispras.ru/sedna/progguide/ProgGuide.html`, 2007. Institute of System Programming of the Russian Academy of Sciences.

[NTH+05] S. Nakajima, J. Tatemura, Y. Hino, Y. Hara, and K. Tanaka. Discovering important bloggers based on analyzing blog threads. In *Procs. of 2nd Annual Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics*, page 8, 2005.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, Brisbane, Australia, 1998.

[Rai05] Lee Rainie. The state of blogging. Available online, at `http://www.pewinternet.org/pdfs/PIP_blogging_data.pdf`, January 2005.

[RZC03] Awais Rashid, Roberto Zicari, and Akmal B. Chaudri. *XML Data Managmnet: Native and XML-Enabled Database Systems*. Addison Wesley Professional, 2003.

[Sif08] David Sifry. The state of the live web, april 2007. Available online, at `http://www.sifry.com/alerts/archives/000493.html`, April 2008.

[Sta01] Kimbro Staken. Introduction to native xml databases. Available online, at `http://www.xml.com/pub/a/2001/10/31/nativexmldb.html`, October 2001.

[THM07] Mohammad A. Tayebi, S. Mehdi Hashemi, and Ali Mohades. B2rank: An algorithm for ranking blogs based on behavioral features. In *WI '07: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, pages 104–107, Washington, DC, USA, 2007. IEEE Computer Society.

[Tor05] James Torio. Blogs: A global conversation. Master's thesis, Syracuse University, 2005.

[Ups05] Trystan Garrett Upstill. *Document ranking using web evidence*. PhD thesis, The Australian National University, 2005.

[W3C06] W3C. Extensible markup language (xml) 1.0 (fourth edition). Available online at: `http://www.w3.org/TR/REC-xml/`, August 2006.

[W3C07] W3C. Xml query test suite. Available online, at: `http://www.w3.org/XML/Query/test-suite/`, July 2007.

[Wes04] Markus Westner. Weblog service providing: Identification of functional requirements and evaluation of existing weblog services in

german and english languages. Master's thesis, UNITEC Institute of Technology, 2004.

[wik08] Wikipedia, the free encyclopedia. `http://www.wikipedia.org`, 2008.

List of Figures

Abstrakt

Blogy sú neustále rastúcim sociálno-ekonomickým fenoménom ovplyvňujúcim mnohé aspekty nášho života. Menia pohľad na novodobú žurnalistiku, výrazne začali konkurovať tradičným periodikám na Internete, ktoré majú svoju oporu aj v printových médiách. Denne vznikajú tisícky nových blogov a objem blogosféry, celého univerza blogov, sa zdojnásobuje každých päť mesiacov. Tento veľký objem dát, podobne ako celý Web, obsahuje články rôznych kvalít; od vysoko cenných až po bezcenný spam. Klasické algoritmy pre hodnotenie kvality webstránok ako sú PageRank a HITS sa ukazujú ako nedostatočné, lebo nevyužívajú dodatočné informácie – charakteristiky blogov. V posledných rokoch sa z tohto dôvodu prebiehajú rôzne výskumy zamerané na analýzu blogov a návrh hodnotiacich algorimtov, ktoré by umožnili triediť ich podľa kvality, či popularity.

Jazyk XML sa stal abecedou webových jazykov. Je to jednoduchý, ľahko rozšíriteľný jazyk vhodný na výmenu a ukladanie štruktúrovaných dát, ktoré sa uplatňujú v rôznych aplikáciach (medicína, legislatíva, personalistika, katalógy). Výhodou jazyka XML je textový formát, dobre čitateľný pre ľudí i stroje.

Uchovávanie údajov v XML nie je nová myšlienka. Uskutočniť ju efektívne v súborovom systéme sa však dá iba nad malou množinou dát z dôvodu neprítomnosti indexov, ktoré by zrýchlili pomalé prehľadávanie rozsiahlych XML dokumentov. Problémom je tiež neexistencia aparátu na modifikáciu XML skombinovateľná s dopytovacími jazykmi pre jazyk XML. Ako riešenie týchto problémov sa ponúka nie veľmi známy a pomerne nový koncept – XML databázy. XML databáza je špecialne určená na ukladanie a manipuláciu s XML dokumentami. Podporuje indexovanie, transakcie a kultivované nástroje na úpravu dát.

Náplňou tejto diplomovej práce je skĺbiť výskum v oblasti XML databáz a hodnotiacich algoritmov pre blogy. Za týmto účelom bol zhotovený plne funkčný komunitný blog portál `blog.maftyz.sk`*, na ktorom boli aplikované výsledky výskumu. Portál beží v súčasnoti na natívnej XML databáze Sedna vyvinutej pri Ruskej Akadémii vied a získava si čím ďalej väčšíi záujem verejnosti. V apríli 2008 mal 150 registrovaných používateľov a 600 publikovaných príspevkov. Aj vďaka týmto používateľom sa dala overiť úspešnosť autormi upravenej verzie EigenRumor algoritmu, v ktorej boli zohľadnené špecifické*

podmienky a výhody tohoto portálu. Rozšírenie ponúknuté v tejto práci využíva zhustený graf príspevkov. Vďaka tejto inovácii po aplikovaní iteratívneho Eigen-Rumor algoritmu dosahuje hodnotenie, ktoré lepšie odráža názor a smerovanie komunity.

Kľučové slová: blogy, XML, XML databázy, hodnotiace algoritmy