A REVIEW PAPER ON

DATABASE MANAGEMENT SYSTEM ANDINFORMATION RETRIEVAL

Authors of Original Research Paper: B. Kiran Kumar, S. Durga Prasad, P M Manohar, KVVS Satya Prakash, M. Chiranjeevi, K. Venkat Kiran

Published in: (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (2), 2012,3632-3637

Research Paper Reviewer/s:

Maheshwari Vidushi
Student**,** Global Public School, Kota-Rajasthan,India -324009
E-Mail: vidushi.maheshwari12@gmail.com

Kumar Ashutosh
Asst. Professor,
School of Commerce and Management,
Career Point University, Kota-Raj. IN.
E-Mail: ashutoshk.rao@gmail.com

ABSTRACT- *In this paper, researchers attempt to bring information retrieval back to database management systems. They have proposed using commercial DBMS as backend to existing full text search engines. Therefore, achieving this search engines directly gain more robustness, scalability, distribution and replication features provided by DBMS. Basically, researchers discussed about the database management system (DBMS) and Information retrieval engagement. Where, it was driven by the increasing in functionality requirements that modern full text search engines have to meet. However, database management systems (DBMS) are not capable of supporting such flexibility. And, with the increase of data to be indexed and retrieved and the increasing heavy workloads, modern search engines suffer from Scalability, reliability, distribution and performance problems. Researchers tried to project a new and simple way for integration and compare the performance of our system to the current implementations based on storing the full text index directly on the file system.*

Keywords: Full text search engines, DBMS IRS, Lucen, performance evaluation, DBMIRS, scalability.

1 Introduction:

The initialization of this research paper basically pertains about the next level application programming where researchers discussed about the systems like office automation systems and other applications need interfaces to for integration of data bases which collaborated with classical data manipulation with management and retrieval of textual ("unformatted") data. Researchers talked about the relational data model, where it is widely accepted as a high-level interface to classical ("formatted") data management. It turns out, however, to be inconvenient for handling even simple data structures as commonly used in information retrieval systems. To encounter this shortcoming, researchers have proposed an extension of the relational model by allowing Non-First Normal Form (NF2) relations. Researchers also summarize the extensions of the relational algebra, with main emphasis on the new "nest" and "unnest" operations which transform between first normal form relations and the NF2 ones. Basically, researchers diagnosed that the database management systems (DBMS) and information retrieval systems (IRS) were separated in research and development and different products have been developed for either purpose. Meanwhile, there is a trend towards a single integrated system for data base management and information retrieval called DBMIRS -because of the following reasons: Many applications need a DBMIRS. Examples are patients' data within hospital information Systems, laboratory document administration, pharmaceutical data bases, and library Information systems, and with growing awareness office information systems. A characteristically feature of these applications is the fact that it is necessary to combine text management and retrieval with usual formatted data manipulation. Therefore, a single user interface is necessary. Most

commercial database management systems offer basic phonetic full text search functionality. For example, Oracle has a module called Oracle Text.[1] Yet, seeking to add more functionality and intelligence to their search capabilities, many commercial applications use third party specialized full text search engines instead. There are several commercial products on the market. But certainly Lucene[2] is the most popular open-source product at the moment. It provides searching capabilities for the Eclipse IDE,[3] the Encyclopedia Britannica CD-ROM/DVD, FedEx, New Scientist magazine, Epiphany, MIT's Open-Courseware[4] and so on. All search engines build an index of the data to be retrieved in user queries. The index is always stored in the file system on disk and can be loaded at startup in the memory (optional in Lucene) for faster querying. However, this is not feasible for large indices due to memory size limitations. So, the standard storage usually remains the file system of the disk. Reliability becomes also a problem. The possibility of corrupting the whole index during a system crash is much higher than losing the data in a database after a similar crash. Restoring a defected index might also take several hours thus complicating the situation even further. The search engine must manage its read and write locks by itself as well. Distributing the index among several sites and providing efficient mirroring techniques is becoming an important issue to large scale search engine projects such as Nutch.[5]They have propose using current DBMS as backend to existing full text search engines as opposed to either reimplementing full text search engine functionality into DBMS or re-implementing core DBMS features into search engines. As a case study, they have used the open-source Lucene and MySQL without loss of generality. To make more prominent and realistic they have used real world data extracted from an electronic marketplace and simulate real world workload traces in order to demonstrate that the overall system throughput and query response time do not suffer with the introduction of DBMS as a backend with their inherent overhead. Where the spectrum of basic infrastructural facilities offered by DBMS the rest of the paper is organized as follows:

- Section 2 provides a background on full text search engines.
- Section 3 presented the proposed system integration.
- Section 4 contains the results of researchers' performance and evaluation.
- Section 5 concludes the entire research paper.

2 Background on Full Text Search Engines

As previously mentioned, that this section pertains the background regarding the text search engines. Where, it discussed about the typical features of search engine, architecture of the full search engine and in the last it discussed about the typical operations of the search engine.

2.1 Typical Features

In this topic researchers project about the full text search engines, that it does not care about the source of the data or its format as long as it is con- verted to plain text. Text is logically grouped into a set of documents. The user application constructs the user query which is submitted to the search engine. The result of the query execution is a list of document IDs which satisfy the predicate described in the query. The results are usually sorted according to an internal scoring mechanism using fuzzy query processing techniques.[6] The score is an indication of the relevance of the document which can be affected by many factors. The phonetic difference between the search term and the hit is one of the most important factors. Some fields are boosted so that hits within these fields are more relevant to the search result as hits in other fields. Also, the distance between query terms found in a document can play a role in determining its relevance. E.g., searching for "John Smith", a document containing "John Smith" has a higher score than a document containing "John" at its beginning and "Smith" at its end. Furthermore, search terms can be easily augmented by searches with synonyms. E.g., searching for "car" retrieves documents with the term "vehicle" or "automobile" as well. This opens the door for ontological searches and other semantically richer similarity searches.

---

[1]Oracle Text. An Oracle Technical White Paper,http://www.oracle.com/technology/products/text/pdf/10gR2text_twp_f.pdf. (2005).

[2]Apache Lucene, http://lucene.apache.org/java/docs/index.html.

[3]B. Hermann, C. Muller, T. Schafer, and M. Me- zini: Search Browser: An efficient index-based search feature for the Eclipse IDE, Eclipse Technology exchange workshop (eTX) at ECOOP (2006).

[4]MIT Open Courseware, MIT Reports to the President (2003–2004).

[5]Nutch home page, http://lucene.apache.org/nutch/

[6]D. Cutting, J. Pedersen: Space Optimizations for Total Ranking, Proceedings of RIAO (1997).

2.2 Architecture

This sub-section illustrated the architecture of the full text search engine mentioned in in Fig. 1, at the heart of a searchengine resides an index. An index is highly efficient cross-reference lookup data structure. In most search engines, a variation of the well-known in- verted index structure is used.[7] An inverted index is an inside-out arrangement of documents such that terms take center stage. Each term refers to a set of documents. Usually, a B+-tree is used to speed up traversing the index structure.

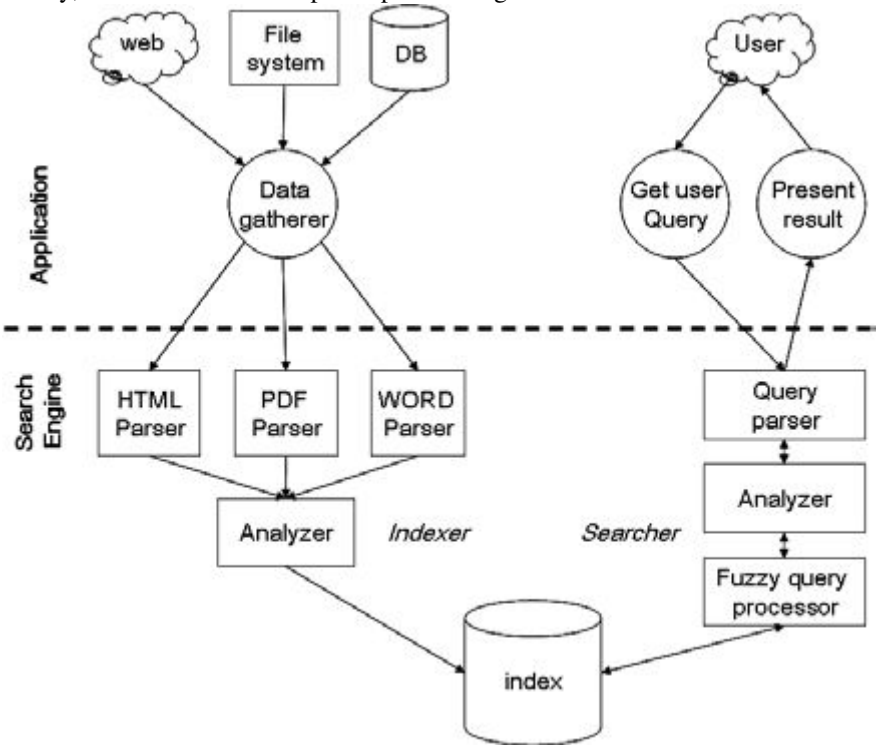

Figure 1: Architecture of a full text search engine

As researchers demonstrated that the indexing process begins with collecting the available set of documents by the data gatherer. The parser converts them to a stream of plain text. For each document format, a parser has to be implemented. In the analysis phase, the stream of data is tokenized according to predefined delimiters and a number of operations are performed on the tokens. For example, the tokens could be lowercased before indexing. It is also desirable to remove all stop words. Additionally, it is common to reduce them to their roots to enable phonetic and grammatical similarity searches.

The search process begins with parsing the user query. The tokens and the Boolean operators are extracted. The tokens have to be analyzed by the same analyzer used for indexing. Then, the index is traversed for possible matches in order to return an ordered collection of hits. The fuzzy query processor is responsible for defining the match criteria during the traversal and the score of the hit.

2.3 Typical Operations

In this sub-section researcher mentioned about 3 main search engine operational issues, these are as follows:

2.3.1 Complete index creation

This operation occurs usually once. The whole set of documents is parsed and analyzed in order to create the index from scratch. This operation can take several hours to complete.

2.3.2 Full text search

This operation includes processing the query and returning page hits as a list of document IDs sorted according to their relevance.

2.3.3 Index update

This operation is also called incremental indexing. It is not supported by all search engines. Typically, a worker thread of the application monitors the actual inventory of documents. In case of document insertion, update, or deletion, the index is changed on the spot and its content is immediately made searchable. Lucene supports this operation.

[7]D. Cutting, J. Pedersen: Optimizations for Dynamic Inverted Index Maintenance, Proceedings of SIGIR (1990).

3 Proposed System Integration

In this section, researchers basically proposed the system segmentation and logical integration. Where they have considered the index file as basic building block and store it in the MySQL database and mapped the logical directory existing database. To project this idea researcher have segmentized it in two sub-sections as follows:

3.1 Architecture

In this sub-section, they have divided the Lucene index into several segments, and the data in each segment is spread across several files. Each index file carries a certain type of information. The exact number of files that constitute a Lucene index and the exact number of segments vary from one index to another and depend on the number of fields the index contains. The internal structure of the index file is public and is platform independent.[8] This ensures its portability.

Researchers considers the index file as primarily building block and procured it in the MySQL database as illustrated in Fig. 2. The set of files, i.e., the logical directory, is mapped to one database relation. Due to the huge variation in file sizes, they have divide each file into multiple chunks of fixed length. Each chunk is stored in a separate tuple in the relation. This leads to better performance than storing the whole file as CLOB in the database. The primary key of the tuple is the filename and the chunk id. Other normal file attributes such as its size and timestamp of last change are stored in the tuple next to the content. Now, they have provided standard random file access operations based on the above-mentioned mapping. Using this simple mapping, researchers takecares of the format of the public index file and present a simple yet elegant way of choosing between the different file storage media (file system, RAM files, or database).

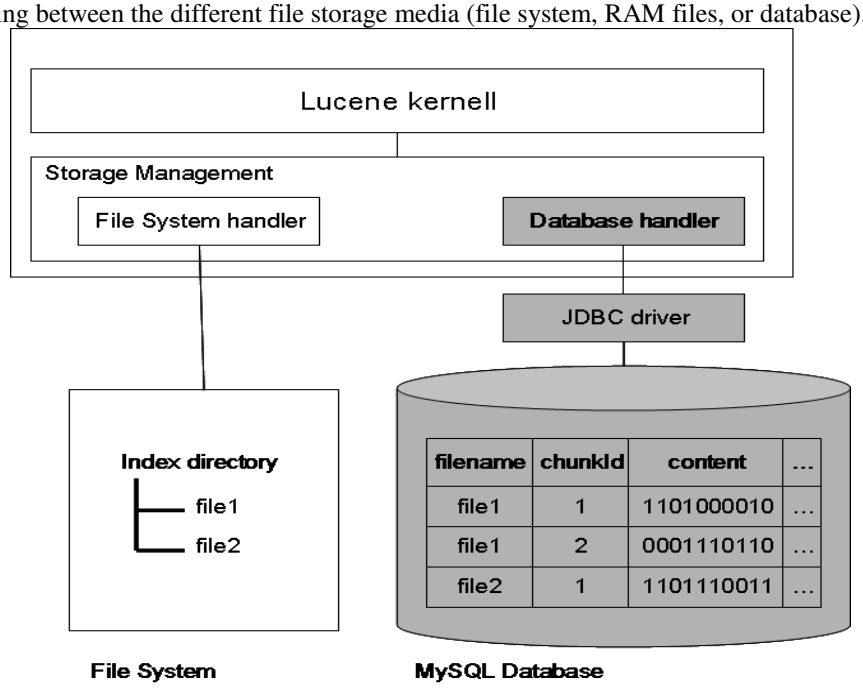

Figure 2: Integrating Lucene index in MySQL database

3.2 System Design

This sub-section discussed about the systematic design of the UML class diagram of the store package of Lucene after modification, illustrated in Fig. 3, and they have included the relevant classesonly. The newly introduced classes are grayed. Directory is an abstract class that acts as a container for the index files. Lucene comes with two implementations for file system directory (FS-Directory) and in-RAM index (RAM Directory). It provides the declaration of all basic file operations such as listing all file names, checking the existence of a file, returning its length, changing its timestamp, etc. It is also responsible for opening files by returning an Input Stream object and creating a new file by returning a reference to a new instance of the Output Stream class. After that, researchers provided a database specific implementation, DB-Directory, which maps these operations to SQL operations on the database.

[8]Apache Lucene - Index File Formats, http://lucene.apache.org/java/docs/fileformats.html.

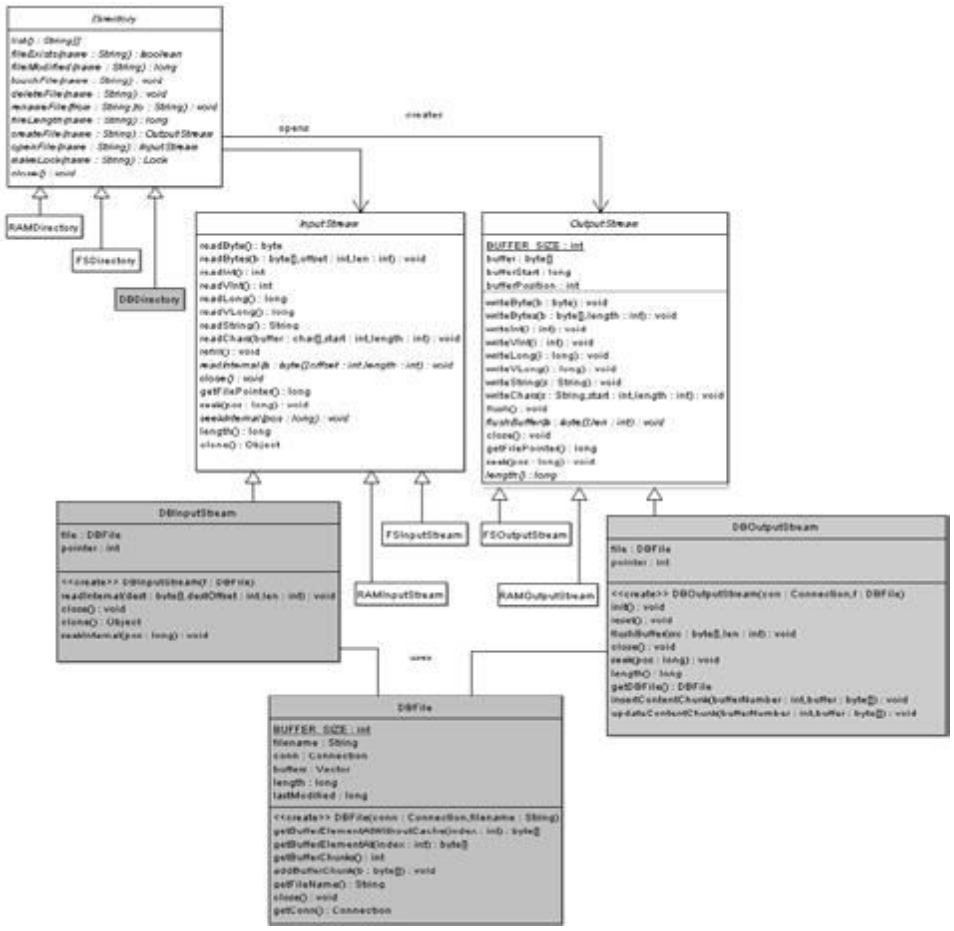

Figure 3: UML class diagram of the store package after modification.

Both Input Stream and Output Stream are abstract classes that mimic the functionality of their java.io counterparts. Basically, they implement the transformation of the file contents into a stream of basic data types, such as integer, long, byte, etc., according to the file standardized internal format.[9]Here as we can see that, researcher have identifies the Actual reading and writing from the file buffer remain as abstract method to decouple the classes from their physical storing mechanism. Similar to FS Input Stream and RAM Input Stream, they have provided the database dependent implementation of the read Internal and seek Internal methods. Moreover, the DB Output Stream provides the database specific flushing of the file buffer after the different write operations. Other buffer management operations are also implemented.

Both DB Input Stream and DB Output Stream use the central class DB File. A DB File object provides access to the correct file chunk stored in a separate tuple in the database. It also provides a clever caching mechanism for keeping recently used file chunks in memory. The size of the cache is dynamically adjusted to make use of the available free memory of the system. The class is responsible for guaranteeing the coherency of the cache.

4 Performance Evaluation

Now this section discussed about the performance & evaluation of the researchers proposed neutralized version of a real electronic marketplace. The index is built over the textual description of more than one million products. Each product contains approximately 25 attributes varying from few characters to more than 1300 characters each. They have developed a performance evaluation toolkit around the search engine as illustrated in Fig.4.The workload generator composes queries of single terms, which are randomly extracted from the product description. It submits them in parallel to the application. The product update simulator mimics product changes and submits the new con- tent to the application in order to update the Lucene index. The application consists of the modified Lucene kernel supporting both file system and database storage options of the full text index. The application under test manages two pools of worker

[9]Apache Lucene - Index File Formats, http://lucene.apache.org/java/docs/fileformats.html.

threads. The first pool consists of searcher threads that process the search queries coming from the workload generator. The second pool consists of index updater threads that process the updated con- tent coming from the product update simulator. The performance of the system is monitored using the performance monitor unit.

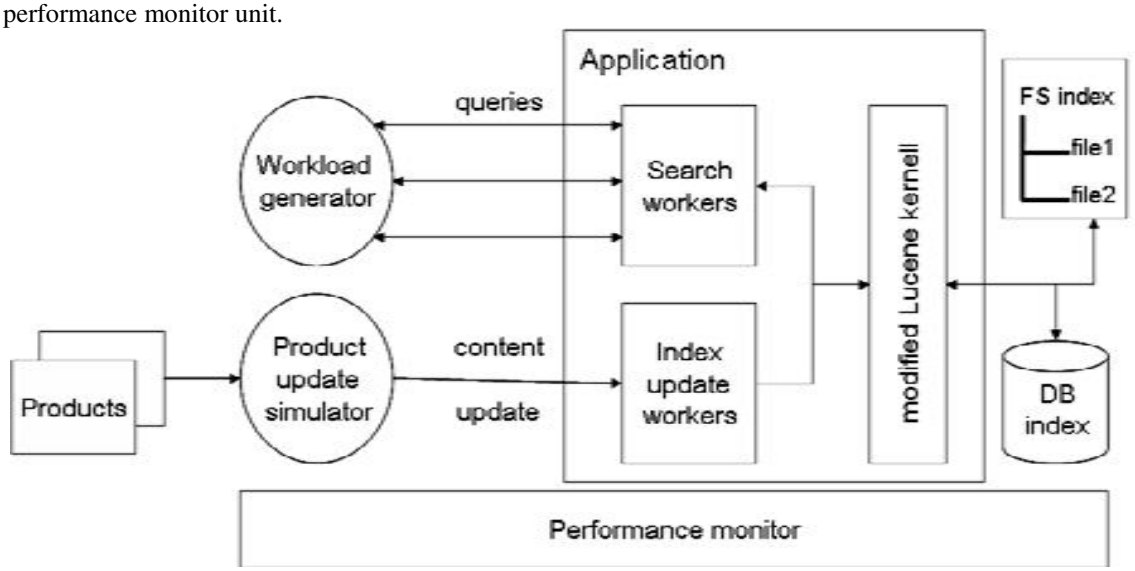

Figure 4: Components of the performance evaluation toolkit.

4.1 Input Parameters and Performance Metrics

In this sub-section, researchers have selected the maximum number of fetchedhits to be 20 documents. This is a reasonable assumption taking into consideration that no more than 20 hits are usually displayed on a web page. The number of search threads is varied from 1 to 25 enabling the concurrent processing of 25 search queries. Due to locking restrictions inherent in Lucene, researchersconfined experiments to maximum one index update thread. They have also introduced a think time varying from 20 to 100 milliseconds between successive index update requests to simulate the format specific parsing of the updated products.

In all the experiments, they have supervised the overall system throughput in terms of conducted:

- Searches per second
- Index updates per second.

Similarly, from the moment of submitting the request till receiving the result, researchers also monitored the response time of:

- The Searches
- The Index Updates

4.2 System Configuration

To perform these experiments,researchers have used a dual core IntelPentium 3.4 GHz processor, 2 GB RAM 667 MHz and one hard disk having 7200 RPM. Where they have calculated the following:

- Data access time of 13.2 micro-seconds
- Seek time of 8.9 micro-seconds
- Latency of 4 micro-seconds

In these experiments, researchers have used the Windows XPoperating system, with JDK 1.4.2, MySQL version 5.0, JDBC mysql- connector version 3.1.12, and Lucene version 1.4.3.

4.3 Experiment Results

Now this section basically deals with the actual results of the experiemnts and their respective evaluations. Here, the performance evaluation considers the mainOperations:

- Complete index creation
- Simultaneous full text search over single terms under various workloads
- In parallel - performing index up- date as product data change

These experiments were conducted for the file system index and the data- base index. Where researchers have dropped the RAM directory from our consideration, since the index under investigation is too large to fit into the 1.5 GB heap size provided by Java under Windows.

4.3.1 Complete Index Creation

To develop the complete index from scratch on the file system,it takes about 28 minutes. Where they have found that the best way to create the complete index for the database is to first create a working copy on the file system and then to migrate the index from the file system to the database using a small utility that we

developed to migrate the index from one storage to the other. This migration takes 3 minutes 19 seconds to complete. Thus, the overhead in this one-time operation is less than 12%.

4.3.2 Full Text Search

In these experiments, researchers have fluctuated the numbers of search threads from 1 to 25 concurrent worker threads and compare the system throughput, illustrated in Fig. 5, and the query response time, illustrated in Fig. 6, for both index storage techniques.

As a result, researchers have found that the performance indices are enhanced by a factor > 2. The search throughput jumps from round 1,250,000 searches per hour to almost 3,000,000 searches per hour in proposed system. The query response time is lowered by 40% by decreasing from 0.8 second to 0.6 second in average. This is a very important result because it means that there is an increase in the performance and take the robustness and scalability advantages of database management systems on top in proposed system.

5 Conclusion & Future Work

In this case study cum experiment, researchers provide a simple system integration of Lucene and MySQL without loss of generality. They have developed the effective performance evaluation toolkit and conduct several experiments on real data of an electronic marketplace. The results show that with the help of these experiments, they have reachedtowards the comparable system throughout and the response times of typical full text search engine operations to the current implementation, which stores the index directly in the file system on the disk. Researchers also stated that, in further experiments and cases, they have reachedeven much better results which mean that they have leadthe robustness and scalability of DBMS on top. And they have also planned on mapping the whole internal index structure into database logical schema instead of just taking the file chunk as the smallest building block. This will solve the restrictive locking problem inherent in Lucene and will definitely boost overall performance. As a future scope and further experimentational perspective they have also planned on expending the achieved performance evaluation toolkit to work on several sites of a distributed database.

References

[1] Oracle Text. An Oracle Technical White Paper,http://www.oracle.com/technology/products/text/pdf/10gR2text_twp_f.pdf. (2005).

[2] Apache Lucene, http://lucene.apache.org/java/docs/index.html.

[3] B. Hermann, C. Muller, T. Schafer, and M. Me- zini: Search Browser: An efficientindex-based search feature for the Eclipse IDE, Eclipse Technology exchange workshop(eTX) at ECOOP (2006).

[4] MIT Open Courseware, MIT Reports to thePresident (2003–2004).

[5] Nutch home page, http://lucene.apache.org/nutch/

[6] D. Cutting, J. Pedersen: Space Optimizations forTotal Ranking, Proceedings of RIAO(1997).

[7] D. Cutting, J. Pedersen: Optimizations for Dynamic Inverted Index Maintenance,Proceedings of SIGIR (1990).

[8] Apache Lucene - Index File Formats, http://lucene.apache.org/java/docs/fileformats.html.

[9] Apache Lucene - Index File Formats, http://lucene.apache.org/java/docs/fileformats.html.