Managing MEDLINE: A Comparison of a Native XML Database System with a Relational Database System

Abstract

The rapid growth of public literature databases like MEDLINE has created the need to efficiently store, retrieve and update the millions of scholarly articles and literature they contain. We believe using alternative database systems like Native XML databases (NXD) will greatly speed up the update process significantly. We used existing and self-developed software packages to parse and load the 2006 release of MEDLINE into two different database systems, namely a NXD (Berkeley DB) and a relational database system (PostgreSQL). The two systems were compared using data collected on loading and parsing times, disk-space utilization and query performance. The NXD offered a significantly faster performance in terms of data parsing and loading times. It was also easier to update and maintain, compared to the relational database system. However, in comparison, the relational database system we tested offered better performance in querying large datasets and was also significantly lower on disk-space utilization.

1. Introduction

The primary source of published scholarly information in the field of biosciences is MEDLINE, which is currently a collection of about 16 million citations maintained by the National Library of Medicine (NLM) [1]. Researchers often refer to this massive information source for various kinds of information. The rapidly increasing rate of new articles submitted to MEDLINE is a major factor contributing to its growth. This poses a challenge to researchers to keep up to date with the latest information. Hence, there is a need for tools that can automate the data loading process and also provide a convenient way to mine specific information from this database. Moreover, due to the large number of people querying the database online, and in an effort to control load on the servers, NLM limits the number of times you can submit a query within a given time frame. Hence, a local instance of the MEDLINE database offers researchers to overcome the limitations of the query options offered by NLM's web interface. It offers greater flexibility in data mining and other applications by allowing individuals to develop their own customized applications to mine information from MEDLINE.

MEDLINE is available for complete download to its licensees at no charge [2]. The files that NLM provides for download are available in the form of several XML (eXtensible Markup Language) formatted files. The 2006 release of MEDLINE consists of a total of 836 compressed files which includes 516 baseline files and 320 update files. The entire distribution requires 8.7 GB of disk space for the compressed files, and 54.1 GB for the uncompressed baseline files [3]. For the purpose of this study, only the 516 baseline files were used to load an instance of MEDLINE. It must also be noted that even though the first 515 of these files contain up to 30,000 citations each, not all citations are of the same size. For instance, the most early of MEDLINE citations lack abstracts, and hence they are of smaller size compared to later citations, that include full text abstracts. Also, not all XML documents contain all defined tags.

1.1 XML Technologies

XML is a free, open-standard markup language, which is flexible, thus allowing its use in a variety of applications. It is called an extensible language because users can create their own custom XML files by modifying, adding or removing tags.

The DTD (Document Type Definition) defines the integral structure of the XML document. It describes the hierarchical arrangement of tags in an XML document. The DTD contains a list of all legal elements and their respective child elements, if any, that may appear in the XML document. An XML document associated with its DTD must comply with the rules and default elements and attributes in the DTD to be considered valid. This serves as a tool to verify syntactic correctness of data.

1.2 XPath & XQuery

The Berkeley DBXML [4] provides XQuery based access to query the database for useful information. XQuery uses path expressions to navigate through the XML document and it uses predicates as constraints to filter the required data. Since data in Native XML Databases (NXDs), is stored as XML documents, it is therefore possible to use the XPath and XQuery data models to construct queries. XQuery comprises of a combination of one or more XPath expressions, arranged according to the FLWOR expression.

FLWOR [12] is an acronym for "For, Let, Where, Order by, Return".
The **for** clause selects all elements under a particular element into a specified variable.
The **where** clause selects only those elements that satisfy a constraint.
The **order by** clause defines the sort-order.
The **return** clause specifies what should be returned.

All NXDs, like the Berkeley DB XML, Xindice [13], eXist [14], etc. store data as XML formatted files, called documents. These documents are stored in a designated container. Each container is capable of storing XML files, called 'documents' in the database. Comparing this system to a relational database system, a container, defined by a specified DTD functions as a table, which has an innate structure defined by its fields, whereas each XML document in a container may be compared to a single row in a relational database.
The XML format despite being a flat file, already has an integral structure, as defined by its DTD; therefore, there is no need to create a database schema, as in relational database systems. The NLM provides a DTD for MEDLINE that defines how the data are represented in the distributed XML files [7]. Berkeley DB XML has an optional feature that allows for validation of each XML document, as it is being loaded into the container. For the purpose of this study, validation was not performed on the XML documents, in order to minimize the loading time.
For the original XML files to be stored in a relational database, extensive amount of parsing needs to be performed. Native XML Databases (NXDs) use an XML document as its fundamental unit, comparable to a row in a relational database. Loading and storing data in the XML format will tremendously ease the loading of data into the database, and will considerably reduce the loading time by eliminating the parsing process. This is especially useful, since NLM provides frequent updates for MEDLINE, sometimes approximately 3 to 4 times a week. Hence, the database needs to be updated frequently to incorporate the updated information. Moreover, since a NXD will preserve the XML format and allow storing the original XML files without any semantic modifications, it will also eliminate the chance of errors or misrepresentation of data.

In this study, we investigate the use of a NXD system (Berkeley DBXML) [4] to host MEDLINE, and study the advantages and possible disadvantages of using such a database format over a traditional relational database system (PostgreSQL) [5] by performing a comprehensive comparative analysis.

The rest of the paper is organized as follows: In Section 2, we give the implementation of both the relational and Native XML databases and describe the process involved to load and store MEDLINE. In Section 3, we present results and discuss the performance comparison of the databases. In Section 4, we discuss related work and finally offer our conclusions in Section 5. The code for our implementation can be found at: http://www.mscs.mu.edu/~praveen/Research/Medline/

2. Implementation

2.1 Relational Database System

For the relational database implementation, we used a modified version of the tool developed and provided by Oliver et al. [6]. It was modified to work with PostgreSQL, whereas the database schema and SQL queries remained unchanged.

2.2 Native XML Database System

The files distributed by NLM are of two kinds, baseline and update files. Each baseline XML file contains upto 30,000 MEDLINE citations [3]. Each XML file contains the root node <*MedlineCitationSet*> with several child nodes called <*MedlineCitation*>, one for each citation or PMID (PubMed ID) (Figure 1). The maximum depth of an XML document that contains all possible tags, as defined by the MEDLINE DTD is 92. In order to store each citation as an individual XML document in the NXD, we developed the *MedlineChunker* program, implemented in Perl, that processes each file by chunking it into individual XML documents, one, for each citation.

```
<MedlineCitationSet>

    <MedlineCitation>
        <PMID>1234</PMID>
        <Article>

    <Author>MiloMinderbinder</Author>
                    .
                    .
                    .
        </Article>
    </MedlineCitation>

    <MedlineCitation>
        <PMID>5678</PMID>
        <Article>

<Author>StiltonCheese</Author>
```

Figure 1: Structure of an XML formatted uncompressed baseline file from MEDLINE: Each distributed baseline XML file has a <MedlineCitationSet> root node, with

several <MedlineCitation> child nodes, one for each MEDLINE citation.

MedlineChunker works by iterating through each compressed file, one at a time, uncompressing it in memory, removes the root tag, and extracts each individual record as data under the <*MedlineCitation*> node using regular expressions, and loads it as a document into the Berkeley DB XML (Figure 2). This method preserves the integral structure of data, as it is represented in the original XML file. Each document in the NXD has a <PMID> node that defines the PubMed ID of the document and works as the unique identifier in the database. The entire process is carried out in memory, in an effort to increase efficiency by avoiding writing dump files, then loading them from disk.

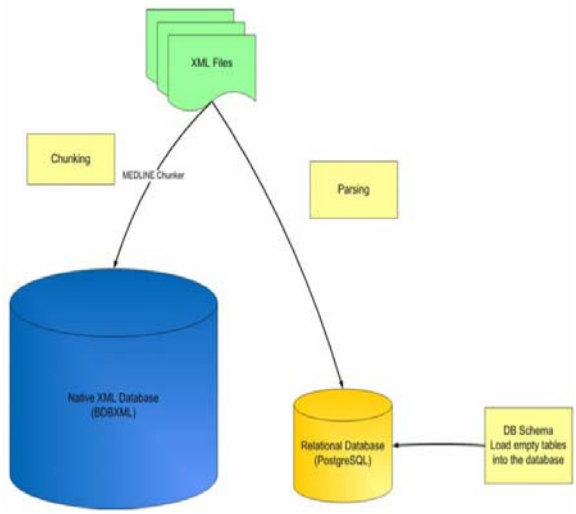

Figure 2. MEDLINE NXD chunking and loading process: In Step 1, the MedlineChunker software uncompresses each MEDLINE distribution file, then breaks up (chunks) each file into individual citations, in the form of several XML documents. In Step 2, the XML documents are loaded into the container.

2.3 Hardware configuration and system requirements

We used a SUN Solaris 10, 750-Ghz dual-processor system, with 4.5 GB of random access memory (RAM). It had a SCSI 3 connected RAID level 5 array with five hard disks, each with a capacity of 400 GB. PostgreSQL 8.0.3 [5] was used to host the relational database and Berkeley DB XML 2.2.13 [4] for the NXD system. Perl 5.8.4 and the Sleepycat DBXml module were utilized to chunk and load the MEDLINE files for the Berkeley DBXML version. Java 1.5.0_01 was used to run a modified version of the Java MedlineParser package [6] to load MEDLINE into PostgreSQL.

3. Results and Discussion

This section describes the comparisons drawn on the two database types on the basis of loading time and disk space utilization as well as the querying times. This is followed by the comparison of query structures and description of the translation process to derive an XQuery for each corresponding SQL query statement.

3.1 Loading time and disk space utilization

Loading time for the NXD was significantly faster and the entire 2006 release of MEDLINE took 48.25 hours to load, whereas the relational database took 92.70 hours to parse and load the data into the database (Table 1). Loading the NXD is much faster, because it does not involve any parsing, and stores the entire XML document as a record, 'as-is'. However, loading into a relational database involves parsing to remove XML tags, which is relatively a very time-consuming process. We also compared loading time using different test sets, each containing a different number of MEDLINE baseline files (Figure 3). The loading of data in PostgreSQL was relatively linear as compared to DBXML. The initial bend in loading time in Berkeley DBXML can be attributed to a lack of abstracts in earlier MEDLINE files, while later MEDLINE files contain full text abstracts.

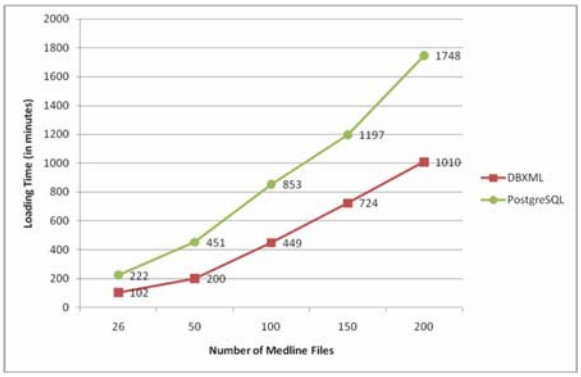

Figure 3. Loading time for Berkeley DBXML and PostgreSQL: Loading time comparison testing of DBXML and PostgreSQL implementations using sets of 26, 50, 100, 150 and 200 baseline files. The slight bend in the DBXML implementation can be attributed to a lack of abstracts in early MEDLINE records, and thereafter, full text abstracts in later citations, where it becomes linear.

The NXD was relatively very high on disk space and occupied 150.3GB and the relational database took

16.8GB of disc space (Table 1). This can be attributed to the fact that for each record, the NXD stores the entire XML document, including all tags, which are repetitive in each document, hence taking up a lot of disc space. On the other hand, as a result of parsing when loading the relational database, all XML tags were eliminated before the data was loaded into the database, thus cutting down on data that needs to be stored in the database.

Database	Language	Input Size	Loading Time	Disc Space
NXD (Berkeley DBXML)	Perl	54.1 GB (516 files)	48.25 Hours	150.3 GB
Relational (PostgreSQL)	Java	54.1 GB (516 files)	92.70 Hours	16.8 GB

Table 1. Loading time and disk space comparison.

3.3 Querying the databases

In order to compare meaningful queries, when using MEDLINE data, we used SQL queries described by Oliver et al. [6]. The SQL queries were each translated into their corresponding XQuery. Table 2 describes a simple SQL statement and the corresponding XPath statement to fetch the Pubmed IDs of all citations in the database. The time taken to perform this query in a relational database took 3.3 minutes, while the NXD counterpart took a long 168 minutes (Table 6). This could be attributed to the fact that Berkeley DBXML stores the entire result set in the memory and then only prints it when required, whereas, a relational database instantly streams the output to the screen without storing the entire result set in the memory.

A	`SELECT pmid FROM medline_citation;`
B	`Collection("mymedContainer.dbxml")/MedlineCitation/PMID/text()`

Table 2. Query 1: Fetch all Pubmed Ids in the MEDLINE database

Query2 (Table 3) fetches the journals that have citations associated with the MeSH term 'Leukemia'. This query took 39.5 minutes in Berkeley DBXML and 7.7 minutes in PostgreSQL (Table 6).

A	`SELECT mc.medline_ta, count(mc.pmid) as num_of_publications FROM medline_citation mc JOIN medline_mesh_heading msh ON mc.pmid = msh.pmid WHERE msh.descriptor_name = 'Leukemia' GROUP BY mc.medline_ta ORDER BY count(mc.pmid) desc;`
B	`let $p:= <tag> {for $x in collection("mymedContainer.dbxml")/MedlineCitation[MeshHeadingList/MeshHeading/DescriptorName = "Leukemia"]/Article/Journal/Title return $x} </tag> for $v in distinct-values($p/Title) order by count($p/Title[.=$v]) descending return concat($v, count($p/Title[.=$v]))`

Table 3. Query 2: Fetch Journals that contain citations that are associated with the MeSH term 'Leukemia' and also the number of such citations for each such journal.

We performed another query (Query 3) that is a modification of the query above to include all citations that have MeSh terms containing the word 'Leukemia' (Table 4). Therefore, such a query would also include terms such as 'Leukemia, Myeloid' and 'Leukemia, Bovine'.

A	`SELECT mc.medline_ta, count(mc.pmid) as num_of_publications FROM medline_citation mc JOIN medline_mesh_heading msh ON mc.pmid = msh.pmid WHERE msh.descriptor name CONTAINS 'Leukemia' GROUP BY mc.medline ta ORDER BY count(mc.pmid) desc;`
B	`let $p:= <tag> { for $x in collection("mymedContainer.dbxml")/MedlineCitation[MeshHeadingList/MeshHeading[contains(DescriptorName,"Leukemia"]/Article/Journal/Title return $x} </tag> for $v in distinct-values($p/Title) order by count($p/Title[.=$v]) descending return concat($v, count($p/Title[.=$v]))`

Table 4. Query 3: Fetch Journals that contain citations that contain the MeSH term 'Leukemia' and also the number of such citations for each such journal.

When comparing querying times for both the databases, it is important to note here that the Berkeley DBXML database could not be indexed for the entire MEDLINE data because in order to index data for a

'contains' query, we need to create an '**edge-element-substring-string**' index, which is an extremely slow and disk-space consuming process on data the size of MEDLINE, as the database stores subsets of each MeSh term. For example, for the term Leukemia, it will store L, Le, Leu, Leuk, Leuke, and so on as individual indexed terms.

A	SELECT 'Berkeley' as institution, count(pmid) as num_of_publications FROM medline citation WHERE CONTAINS(article_affiliation,'"Berkeley"') = 1 AND date_created > 2003 UNION SELECT 'Stanford' as institution, count(pmid) as num_of_publications FROM medline citation WHERE CONTAINS(article_affiliation,'"Stanford"') = 1 AND date_created > 2003;
B	let $x := collection("mymedContainer.dbxml") /MedlineCitation[contains(Article/Affiliation/text(), "Berkeley")] let $y := $x/DateCreated[Year > 2003] let $p := collection("mymedContainer.dbxml") /MedlineCitation[contains(Article/Affiliation/text(), "Stanford")] let $q := $p/DateCreated[Year > 2003] return (concat ("Berkeley", count($y)), concat ("Stanford", count($q)))

Table 5. Query 4: Fetch the number of citations published in the last three years by Berkeley and Stanford.

Table 5 illustrates Query 4, which fetches the number of papers published in MEDLINE by Berkeley and Stanford in the last 3 years. For this particular query, since it was not a 'contains' query, it was easier to index the database for the 'Year' node. Therefore, this query had much better response time compared to other queries. Berkeley DBXML completed this query in 4.3 minutes. PostgreSQL took 3.8 minutes to complete this query (Table 6).

Database	Query 1	Query 2	Query 3	Query 4
DBXML	168	39.5	196	4.3
PostgreSQL	3.3	7.7	5.4	3.8

Table 6. Querying time comparison; Time in minutes.

4. Related Work

In Table 7, we summarize our survey of related research works, which have undertaken performance evaluation of relational databases versus native XML databases across different features: disk space utilization, loading time, query performance and use of bioinformatics data.

Paper Reference	Disk space utilization	Loading Time	Query performance	Use of bioinformatics data
[15]	N	Y	Y	N
[16]	Y	N	Y	N
[17]	N	N	Only XML*	Y
[18]	N	N	Y	N
[19]	N	N	Y	N
Our work	Y	Y	Y	Y

Table 7. Comparison parameters used to evaluate databases to store XML data in various studies. *This study evaluated only native XML and relational databases with XML packages to store XML data, however it involved the use of bioinformatics data as part of the test set.

The use of bioinformatics data for comparison purposes is particularly interesting to biologists and computer scientists who work with similar kind of data. Benjamin Bin Yao et al. [15] describe the XBench XML benchmark and evaluate the relative performance of various DBMSs. Similar to the results in our study, they found native XML databases performed much faster, compared to a relational database (SQL Server) with an XML package. This confirms our belief that native XML databases can be a promising data management standard of the future, with increasingly more data sources releasing their data in the XML format.

Alan Halverson et al. [16] propose a method for storing XML data in traditional relational databases, called Relational Over XML (ROX). Their results show the native XML database requiring more disk space for the same data-set compared to the relational database, which is consistent with our results; however, with data storage becoming increasingly cheaper, this should not be a major concern.

Yi Chen et al. [17] propose a polynomial time streaming algorithm to evaluate XPath queries. It was observed that this method streamlines the query processing time by storing data in a lazy fashion. They test this approach using several native XML databases.

Zhen Hua Liu et al. [18] discuss the Oracle XMLDB XQuery architecture and its capabilities for natively supporting these XQuery operations using SQL/XML standard functions. Hongjun Lu et al. [19] evaluate the

performance of various XML databases including native XML databases and relational databases with XML mapping approaches. They test these databases on the basis of query response times. Their results indicate that the native XML databases by far outperformed the document-dependent relational databases in terms of query processing times.

5. Conclusions

There is a growing need for both computer scientists and biologists to keep an up-to-date version of MEDLINE in their local system. However, using existing tools of relational database system is a daunting task because of the sheer amount of time it takes to load and parse data. Hence, alternative databases such as Native XML Databases (NXDs) are one option to alleviate such problems. Berkeley DBXML was significantly faster for loading data, compared to PostgreSQL. However, the relational database surpassed Berkeley DBXML when it came to querying large sets of complex result sets. The native XML database did not have efficient memory management capabilities, as compared to the relational database, which prevented it from performing SORT and GROUP BY operations on larger data sets. Moreover, it was also noted that conventional relational databases have superior and well developed indexing techniques compared to the native XML database, which did not perform well in contain queries because it is not feasible to index data on the basis of substrings.

We hope our study has given pointers to scientists debating over the use of NXDs versus relational database systems. We also believe by solving the challenges of efficient indexing and memory management techniques for NXDs, these could be powerful tools for efficiently maintaining MEDLINE information.

6. References

1. National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov
2. Leasing data from the NLM, http://www.nlm.nih.gov/databases/leased.html
3. File Names, Record Counts, and File Size for 2006 MEDLINE/PubMed Baseline Database.Distribution, http://www.nlm.nih.gov/bsd/licensee/2006_baseline_med_filecount.html
4. Berkeley DBXML, http://www.oracle.com/database/berkeley-db.xnl/index.html
5. PostgreSQL, http://www.postgresql.org/
6. Oliver, DE., Bhalotia, G., Schwartz, AS., Altman, RB., Hearst, MA.: Tools for loading MEDLINE into a local relational database. BMC bioinformatics, 5:146 (2004)
7. DTDs for NLM Databases, http://www.nlm.nih.gov/databases/dtd/index.html
8. Hulse, N., Rocha, R., Del Fiol, G., Bradshaw, R., Hanna, T., Roemer, L.: The Knowledge Authoring Tool: An XML-based Knowledge Acquisition Environment. Conf Proc IEEE Eng Med Biol Soc, 5:3350-3353 (2004)
9. Mudunuri, U., Stephens, R., Bruining, D., Liu, D., Lebeda, FJ.: botXminer: mining biomedical literature with a new web-based application. Nucleic acids research, 34(Web Server issue):W748-752 (2006)
10. Shaohua, Alex Wang YF., Huey, C., Frank, P,, Barg, U., Adam, F., Raj, L., Sarada, C., Gladys, W., Marc, K., Robert, L. Martino, Calvin A. J.: Performance of Using Oracle XMLDB in the Evaluation of CDISC ODM for a Clinical Study Informatics System. 17th IEEE Symposium on Computer-Based Medical Systems (2004)
11. Seibel PN., Kruger J., Hartmeier S., Schwarzer K., Lowenthal K., Mersch H., Dandekar T., Giegerich R.: XML schemas for common bioinformatic data types and their application in workflow systems. BMC bioinformatics, 7:490 (2006)
12. XQueryFLWOR expressions, http://www.w3schools.com/xquery/xquery_flwor.asp
13. Xindice, http://www.xindice.org
14. eXist, http://exist.sourceforge.net
15. Yao, B., Ozsu, T., Khandelwal, N.: XBench Benchmark and Performance Testing of XML DBMSs. Proceedings of the 20th International Conference on Data Engineering (2004)
16. Halverson, A., Josifovski, V., Lohman, G., Pirahesh, H., Mörschel, M.: ROX: Relational Over XML. Proceedings of the 30th VLDB Conference, Toronto, Canada (2004)
17. Chen, Y., Davidson S., Zheng, Y.: An Efficient XPath Query Processor for XML Streams. Proceedings of the 22nd International Conference on Data Engineering, ICDE (2006)
18. Liu, Z., Krishnaprasad, M., Arora, V.: Native XQuery Processing in Oracle XMLDB, Oracle Corporation.
19. Lu, H., Jiang, H.: What Makes the Differences: Benchmarking XML Database Implementations. ACM Transactions on Internet Technology, Vol. 5, No. 1 (2005)