Semi-structured Data Management in the Enterprise: A Nimble, High-Throughput, and Scalable Approach

David Maluf, David Bell, Naveen Ashish, Chris Knight and Peter Tran
NASA Ames Research Center, Moffett Field, CA 94035, USA
David.A.Maluf@nasa.gov

Abstract

In this paper we describe an approach and system for managing enterprise semi-structured data that is high-throughput, nimble, and scalable. We present the NETMARK system, which provides for a "schema-less" way of managing semi-structured documents. We describe in particular detail the unique underlying data storage approach and efficient query processing mechanisms given this storage system. We present an extensive benchmark evaluation of the NETMARK system and also compare it with related XML management systems. At the heart of the approach is the philosophy of a focus on most common data management requirements in the enterprise, and not burdening users and application developers with unnecessary complexity and formal schemas.

1. Introduction

Searching, extracting, and integrating information from documents, in a simple yet precise manner, is a key requirement in many enterprise-wide information systems applications. The term 'documents' here includes textual documents such as reports and other documents in formats such as MSWord, PDF or others, spreadsheets, presentations in formats such as MS Powerpoint etc.[1] Such information is typically "semi-structured" in that there is some structure in the documents but not exactly a formal structure such as that imposed by a database schema or an XML DTD[20]. While there are indeed systems available for managing semi-structured or unstructured data, such as DocuShare[6] from Xerox, products from companies like Verity[2] and Autonomy[3] and research systems for semi-structured or XML data management[9,11,15], we have designed a system that is significantly more flexible and simple from a user perspective and scalable from an application development perspective. We describe the NETMARK system[13] that the NASA Ames Research Center[4] has developed and that has been used for several NASA[5] enterprise data and project management applications. The focus of this paper will particularly be on the data storage and query processing aspects of NETMARK

A key distinguishing feature of the NETMARK approach (vis-à-vis other semi-structured data or XML data management systems) is that it does not require users (or administrators) to have to formally define the semantics of the data (schemas) in the documents or collections of documents. Structure and semantics information implicit in the document is exploited instead. This leads to a system where sophisticated data integration and *composition* applications can be built without high schema management overheads, in a highly scalable manner.

This paper is organized as follows. In section 2, the basic document query and search paradigm around which the NETMARK approach is centered is described. Section 3 contains the technical details of the approach, focusing first on document storage issues where a unique node structure representation for document storage is presented. This is followed by a description of query processing details. Section 4 provides extensive benchmarking results evaluating the performance of the NETMARK system followed by a comparison of NETMARK with related semi-structured and XML data management systems in Section 5 and a conclusion.

2. Document Querying

The data querying capabilities in NETMARK are centered around the notions of *context* and *content* in documents. Context and Content are notions that

[1] Indeed, 80% of the enterprise data is stored in such unstructured or semi-structured documents, instead of in databases, according to research firm Gartner

[2] www.verity.com

[3] www.autonomy.com

[4] http://www.nasa.gov/centers/ames/home/index.html

[5] http://www.nasa.gov/home/index.html

facilitate perceiving and querying a document by various components (sections and sub-sections) as opposed to treating a document as a single unit. A document typically has an inherent structure, i.e., it can be perceived to be comprised of various distinct sections and sub-sections based on that structure. For instance consider one of the monthly project reports shown in Fig 1. It comprises of various sections such as Report Month, Report Year and Performance Status, and sub-sections such as Technical Status, Schedule Status etc. A context is essentially a section or sub-section within a document. So for instance for the monthly project report document as described above, the Report Year, Org, Performance Status, Tech Mgmt Comment, would all be contexts. The information (for instance the text) within a context is referred to as content.

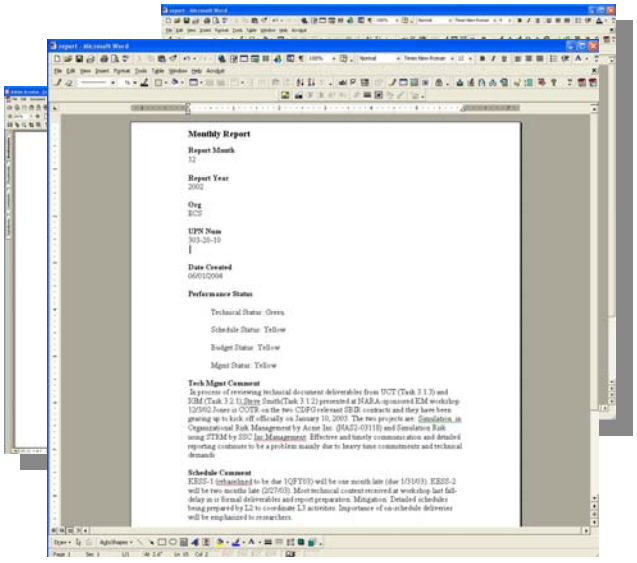

Fig 1. Monthly Report Documents

For instance any text (and figures or tables) in the Schedule Status context (sub-section) is the content *associated with* the Schedule Status context. Context and content thus are associated in pairs, the content being associated with a context. As another example, the contexts for this paper as an example of a document, are sections such as the Abstract, Introduction, Document Querying, etc. The query capabilities in Netmark, built around the notions of context and content, have been developed based on our knowledge of the most common and important queries to documents and semi-structured data in typical enterprise data applications. A key capability is that of *context search*. A context search query, such as "Context=Budget Comment"[6] will return the content portion in the 'Budget Comment' sections (the text in the Budget Comment section) in *all* the documents in a document collection, as illustrated in Fig 2. A context query thus extracts the specified context (section) from all documents and returns it to the user.

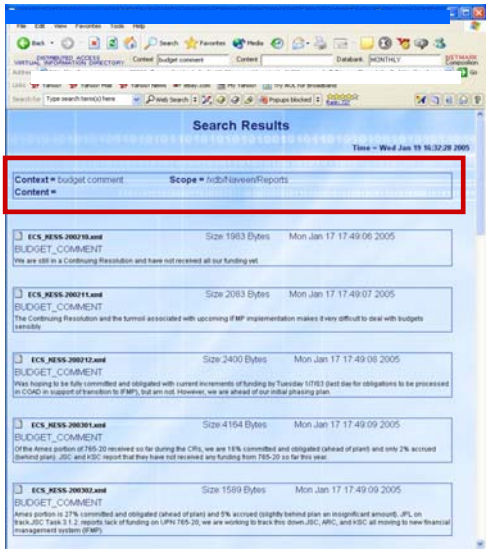

Fig 2. Context Query Results

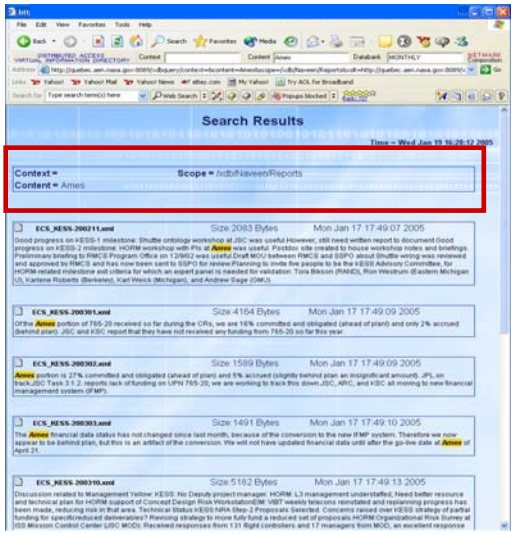

Fig 3. Content Query Results

[6] This is not the precise query syntax and we do not think it essential to use the formal and precise Netmark query syntax here

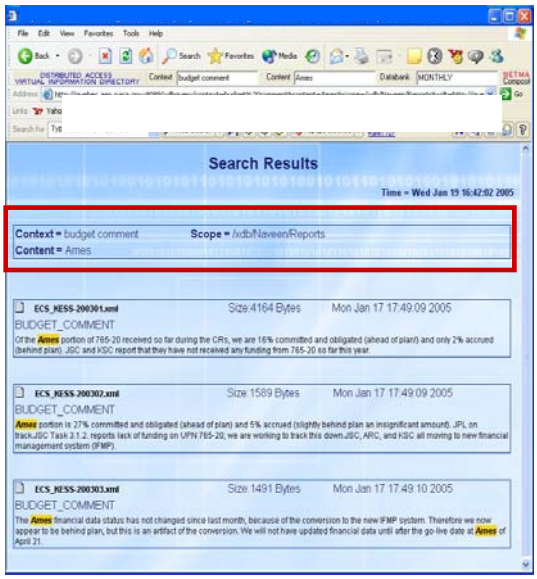

Fig 4. Context+Content Query Results

Users can also specifying *content searches*, which are essentially keyword searches that return all documents containing the specified search terms. For instance, a content query such as "Content=Ames" will return all documents that contain the term 'Ames' anywhere in the document as shown in Fig 3. One can also. combine context and content searches, for instance a query such as "Context=Budget Comment&Content=Ames" returns the "Budget Comment" contexts (sections) of all documents where the term 'Ames' occurs *within the Budget Comment context (section)* as shown in Fig 4.

Thus, as opposed to conventional keyword search systems which treat a complete document as a single unit, NETMARK facilitates searching with respect to specified *contexts* in the documents. The NETMARK query language is a language called XDB Query. XDB Query allows for posing the context and content kinds of queries over XML documents, as illustrated above. We will not go into the query syntax details here but the key features are that context and content search specifications are appended to a URL that is sent to NETMARK. An example of a formal XDB query is:

http://larry.aen.nasa.gov:32080/xdbquery/context=BudgetComment&content=Ames

In this URL we may also specify an XSLT stylesheet which specifies how the results are to be formatted and composed into a new document. Fig 5. provides an illustration of using XDB Query to query the data in NETMARK and then using XSLT to format the results. XSLT transformation is done using the Xalan XSLT processor [19].

The basic query syntax for XDB queries is:

https://<server_address>**/xslt/xdbquery/**{[**context=**<context_keys>]|[**&content=**<content_keys>]}|[**&scope=**<relative_url_to_folder>]|[**&syntax=**{html, xml, ascii}]|[**&xslt=**<relative_url_to_xslt

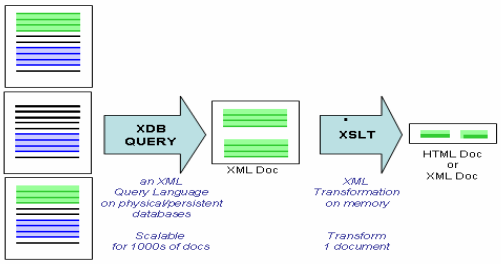

Fig 5. Formatting Results

Context and content based queries provide powerful primitives for querying and integrating data from semi-structured data documents. Also the notions of context and content extend to documents such as presentations (in say Powerpoint) or Excel spreadsheets as well. For instance one might require the 'Budget' sections out of each presentation in a collection of powerpoint presentations and a context query for 'Budget' on that collection would facilitate that. There are other parameters that can be specified in context or content queries such as controlling the maximum number of documents returned, the "depth" of the result items and others but we will not go into those details here. Indeed such query and formatting capabilities have proved to be quite powerful and adequate for quickly developing several large scale applications (entailing significant enterprise data extraction and integration) in the NASA domain. How NETMARK provides simple yet powerful document querying capabilities is described in the following section

3. Technical Details

In this section the technical details of NETMARK are described, in particular the storage and query processing aspects. Note that NETMARK serves as a semi-structured data management system and also provides (integrated) access to legacy data sources in enterprises[13]. The NETMARK system architecture is first presented. Then a unique approach to storing semi-structured data documents is presented followed by a description of the query processing approach.

3.1 Integrated Legacy Data and XML Data Access

The NETMARK system architecture is outlined in Fig 6. below. We will not delve into the complete architectural details in this paper and refer the reader to[13] for those.

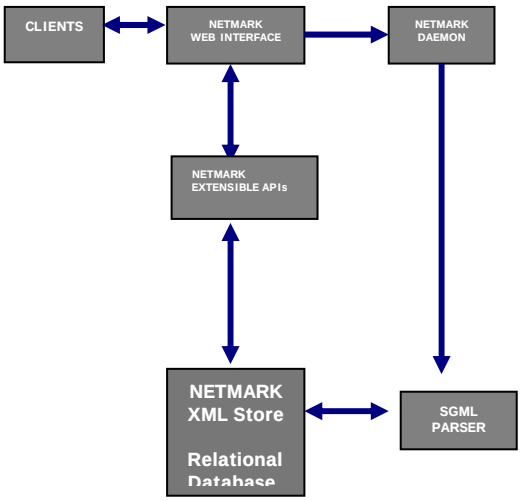

Fig 6. Netmark System Architecture

What is to be noted for this paper is that the underlying data store in NETMARK is a relational database system. The other components shown are those that insert new documents into the NETMARK data store (i.e., the NETMARK DAEMON and the SGML PARSER described in a later sub-section) and the NETMARK APIs for end user clients.

3.2 Data Storage in NETMARK

In the section we describe how semi-structured documents are stored internally in NETMARK. The approach has been to keep the underlying representation simple, yet expressive enough to store hierarchical relationships in documents.

3.2.1 Node Structure Representation

In related systems, for instance XML data management systems that have been built on top of relational database systems [17], we have specified mechanisms for mapping XML DTDs to relational schemas and thus storing XML data in corresponding relational tables. Different XML DTDs are mapped to different sets of relational tables. NETMARK uses a simpler and more flexible approach where the *same* (two) relational tables are used to represent and store the data in *any* semi-structured document. The approach is based on a searchable *node* structure which is the eventual representation for documents that are stored in the system. 'Raw' documents (initially in any format such as Word, PDF etc.) are first converted[7] into XML, which describes the document as decomposed into various sections and sub-sections. The XML representation essentially captures the various contexts and content associated with each context in a document. For instance the XML representation of this paper would be as shown in Fig 7.

```
.....
<Context>Abstract</Context>
    <Content> This paper describes an ... </Content>
<Context>Introduction</Context>
    <Content> Searching, Querying and Integrating ...
</Content>
<Context> Document Querying </Context>
    <Content> A document typically has an inherent ...
</Content>
.......
```

Fig 7. Context and Content Segmentation

We begin by assigning a node to each section or sub-section in a document. A node is basically the 'container' for a context or content in the document. Each node has an assembly of labels or attributes for the node. The attributes for each node include the following:
DOCID: A unique number assigned to the document.
NODEID: A unique identifier for each node.
NODENAME: A descriptive name for the node

[7] The NETMARK system includes converters which automatically convert documents in Word, PDF, Excel and other formats to XML. The converters have been built on top of frameworks such as Apache Jakarta POI (http://jakarta.apache.org/poi/) and JPedal for PDF (http://www.jpedal.org/)

NODETYPE: Identifies the node type, which is one of a small list of mutually exclusive node types.
NODEDATA: The actual content of the node.
PARENTROWID: Contains the ROWID of a parent of the node (if any).
SIBLINGID: Contains the ROWID of a sibling of the node (if any).

Essentially a document gets divided into blocks of context and associated content. A node serves to hold each individual context and content block in a document. For instance, for this paper, for the Introduction context we would have a CONTEXT type node (i.e., where NODETYPE=CONTEXT) for that context. The NODEDATA for this node would be the name of the context i.e., the string "Introduction". We would have another node of the type TEXT where the NODEDATA would be all the text and figures in the Introduction section. This content node would also be a child of its associated context node.

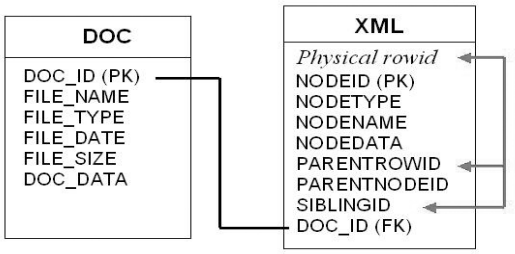

Fig 8. Relational Tables for Node and File Information

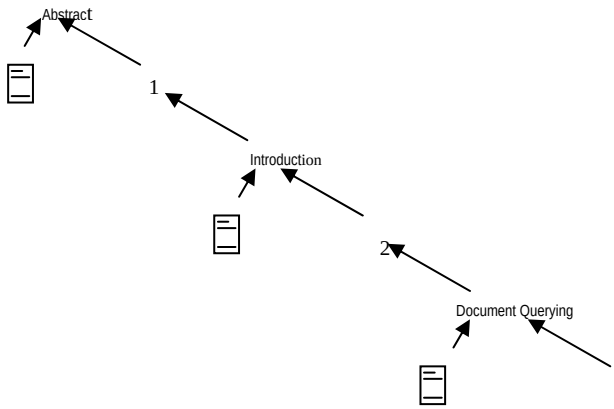

Fig 9. Node Representation and Relationships for this Paper as an Example Document

The tree in Fig 9. shows some of the nodes and their relationships corresponding to the node structure representation of this paper. The arrows denote parent-child relationships between the nodes. We see CONTEXT nodes such as 'Abstract' and 'Introduction'. TEXT nodes capturing content for any context are placed as children of their corresponding context nodes. For instance, the left child of the Abstract context node is the content node (containing the text for the abstract section) for that context, etc. Also, each context node is a child of the context immediately preceding it in the document. The PARENTROWID and SIBLINGID elements in each node are basically pointers that capture the parent child relationships between the various nodes.

To summarize the above, a raw semi-structured data document is first converted to XML where the various contexts and contents in the document are segregated. Each context or content is stored in a node structure and parent child relationships (including context-content association) are captured through pointers between nodes. For each document inserted into NETMARK, document information is stored in the DOC table and various nodes are stored in the XML table, shown in Fig 8. Both the DOC and XML tables shown in Fig 8. are relational tables in the NETMARK underlying relational data store.

3.2.2 ROWIDs for Node Access and Traversal

Each node contains pointers to its parent and sibling nodes. Query processing (described shortly) requires us to retrieve nodes related to a node (such as retrieving the parent or sibling of a node) in a very fast manner. For this NETMARK exploits the features of ROWIDs, a data type in Oracle 9i[1], which is the relational data store over which NETMARK is built. ROWID is an Oracle data type that stores either physical or logical addresses (row identifiers) to every row within the Oracle database. Physical ROWIDs store the addresses of ordinary table records (excluding indexed-organized tables), clustered tables, indexes, table partitions and sub-partitions, index partitions and sub-partitions, while logical ROWIDs store the row addresses within indexed-organized tables for building secondary indexes. Each Oracle table has an implicit pseudo-column called ROWID, which can be retrieved by a simple SELECT query on the particular table. Physical ROWIDs provide the fastest access to any record within an Oracle table with a single read block access, while logical ROWIDs provide fast access for

highly volatile tables. A ROWID is guaranteed to not change unless the rows it references is deleted from the database.

The physical ROWIDs have two different formats, namely the legacy *restricted* and the new *extended* ROWID formats. The restricted ROWID format is for backward compatibility to legacy Oracle databases, such as Oracle 7 and/or earlier releases. For example, the following displays a subset of the extended ROWIDs from a NETMARK generated schema. It is a generalized 18-character format with 64 possibilities each:

AAAAAA | BBB | CCCCCC | DDD

The extended ROWIDs could be used to show how an Oracle table is organized and structured; but more importantly, extended ROWIDs make very efficient and stable unique keys for information retrieval, which will be addressed in the sub-section below on query processing

3.2.3 Mapping from Hierarchical to Relational

Also, object-relational mapping from XML to relational database schema models the data within the XML documents as a tree of objects that are specific to the data in the document [14]. In this model, element type with attributes, content, or complex element types are generally modeled as classes. Element types with parsed character data (PCDATA) and attributes are modeled as scalar types. This model is then mapped to the relational database using traditional object-relational mapping techniques or via SQL3 object views. Therefore, classes are mapped to tables, scalar types are mapped to columns, and object-valued properties are mapped to key pairs (both primary and foreign). This mapping model is limited since the object tree structure is different for each set of XML documents. On the other hand, the NETMARK SGML parser models the document itself (similar to the DOM), and its object tree structure is the *same* for all XML documents. Thus, NETMARK is designed to be *independent* of any particular XML document schemas and is termed to be "*schema-less*".

3.3 Query Processing

The NETMARK keyword-based context and content search is performed by first querying text index for the search key. Each node returned from the index search is then processed based on its designated unique ROWID. The processing of the node involves traversing up the tree structure via its parent or sibling node until the first context is found. The context is identified via its corresponding NODETYPE. The context refers to here as a heading for a subsection within a HTML or XML document, similar to the <H1> and <H2> header tags commonly found within HTML pages. Thus, the context and content search returns a subsection of the document where the keyword being searched for occurs. Once a particular CONTEXT is found, traversing back down the tree structure via the sibling node retrieves the corresponding content text. The search result is then rendered and displayed appropriately.

Query processing in NETMARK differs from related XML data management systems that have been built over relational database systems. The prime reason is that the way XML data is stored (in a relational database) in NETMARK is fundamentally different from the manner it is stored in other XML over relational systems. One of the earlier efforts such as [16] describe techniques

such as *basic inlining* and *shared inlining*, which are schemes for mapping (simplified) XML DTDs to relational schemas without loss of information. A concern from a query processing perspective is that of the cost of processing XML queries with (possibly lengthy) path expressions as such queries lead to multiple joins being performed across the underlying relational tables, which is expensive. The query processing performance under various alternative mapping schemes is evaluated in [16]. In a related system called XTABLES[11], which is also an XML over relational system, a query processing scheme based on an intermediate query representation is described. The XTABLES query processor attempts to maximally harness the power of its underlying relational engine by pushing down most memory and data-intensive computation to the underlying relational engine. A user query, in XQuery, is first converted into an intermediate XML Query Graph Model representation (XQGM). Rewrite optimizations are performed on the XQGM and the data and memory intensive part is pushed down to the relational engine as a single SQL query. A 'tagger run-time module' constructs the XQuery result from the results of the SQL query and returns it to the user.

In NETMARK, we are using ROWIDs (physical address) to traverse between nodes. A ROWID provides the fastest access to a record or corresponding node within a relational table, with a single block read access. Accessing a record based on its physical address ROWID provides an efficient, constant access time C (machine dependent; normally in the millisecond range) that is independent of the number of records or nodes in the database and regardless of maximum node depth within a node structure. The time to respond to a context or content query is thus

approximately proportional to log(N) (first search time) plus a sum of the Cs for each successive search where N is the number of records or nodes.

4. Benchmarking

The NETMARK system is intended for managing large documents and/or large collections of documents. The question arises, as with any other data management system, regarding the performance of NETMARK. In fact performance of a couple of different aspects is of interest. We are certainly interested in query processing performance for various kinds of queries. We are also interested in the performance of the NETMARK pipeline i.e., how efficiently can NETMARK load in documents input into the system. For evaluating query processing, database systems have had a long standing tradition of benchmarking the databases against various standard benchmarks. Benchmarks have also been proposed for XML data management systems[2]. In NETMARK our prime focus has been on queries centered around context and content, although NETMARK does support full fledged Xpath queries over documents as well. For evaluating query processing, we thus focused on evaluating the context and content kinds of queries in NETMARK. We have also compared NETMARK with another XML data management system – Berkeley XML[8]. Also, we have evaluated the performance of the NETMARK pipeline. All the evaluation results are presented below.

4.1 Benchmarking Query Processing

We present below results of benchmarking the performance of NETMARK, including a comparison with Berkeley XML, another industry XML data management system. We tested a variety of queries centered around context and content. The benchmarks were conducted on a Dell Poweredge 1650 server with 2 Intel Pentium III 1.2 GHz processors, with 1.3G of RAM and running RedHat Enterprise LINUX AS Release 3.

4.1.1 Benchmarking Document

A generated XML document with the DTD shown in Fig 10. was used. We have evaluated the query processing performance of both NETMARK and Berkeley XML for various document sizes, namely 100 MB, 50 MB, 25 MB, and 10 MB. A variety of context and content oriented queries for the test document(s) were tested. For context and context+content queries we were able to express equivalent queries (in XQuery) in Berkeley XML. For content queries, even with the XQuery "contains" operator, we cannot pose queries in Berkeley XML with semantics equivalent to the NETMARK/XDB content query.

[8] http://www.sleepycat.com/products/xml.shtml

```
<?xml version="1.0"?>
<!DOCTYPE products [
  <!ELEMENT products (product)+
productU (productZ)+>
  <!ELEMENT product (item, category,
vendor, vendor_2, vendor_3)>
  <!ELEMENT productU (itemU, category,
vendor, vendor_2, vendor_3)>
  <!ELEMENT productZ (itemZ, category,
vendor, vendor_2, vendor_3)>
  <!ELEMENT item (#PCDATA)>
  <!ELEMENT itemU (#PCDATA)>
  <!ELEMENT itemZ (#PCDATA)>
  <!ELEMENT category (#PCDATA)>
  <!ELEMENT vendor (#PCDATA)>
  <!ELEMENT vendor_2 (#PCDATA)>
  <!ELEMENT vendor_3 (#PCDATA)>
]>
```

Fig 10. DTD of Benchmarking Data Document

4.1.2 Results

Benchmarking with single XML document

Test document size = 100 MB

Queries	NET MARK	Berkeley XML	Result Size
Context Query *context=itemZ*	6990 †		4,000 * [441362]
Context Query *context=productZ*	7126		4,000 [9715210]
Context Query *context=productU*	234		1 [1561]
Context Query *context=itemU*	81		1 [540]
Context+Content Query *context=item&content=Lemon Grass*	749		1 [482]
Context+Content Query *context=category & content=Fruits*	720		1 [399]
Content Query 1	685000		[65557376]
Content Query 2	2420		[672]

* Number on first line denotes size in number of XML elements. Number in [] denotes size in characters.
† All times are in milliseconds (ms)

Test document size = 50 MB

Queries	NET MARK	Berkeley XML	Result Size
Context Query *context=itemZ*	3340	54,000	2,000 [220680]
Context Query *context=productZ*	3680	56,000	2,000 4857604
Context Query *context=productU*	218	48,001	1 [1561]
Context Query *context=itemU*	84	49,005	1 [540]
Context+Content Query *context=item&content=Lemon Grass*	707	540007	1 [482]
Context+Content Query *context=category & content=Fruits*	742	540085	1 [399]
Content Query 1	497000	‡	[32778688]
Content Query 2	2190		[674]

Test document size = 25 MB

Queries	NET MARK	Berkeley XML	Result Size
Context Query *context=itemZ*	2100	23,806	1,000 [110340]
Context Query *context=productZ*	2200	21,141	1,000 [2428802]
Context Query *context=productU*	210	18,400	1 [1561]
Context Query *context=itemU*	79	19,995	1 [540]
Context+Content Query *context=item&content=Lemon Grass*	698	99,000	1 [482]
Context+Content Query *context=category & content=Fruits*	692	114,239	1 [399]
Content Query 1	307000	‡	[16389344]
Content Query 2	1950		[675]

Test document size = 10 MB

Queries	NET MARK	Berkeley XML	Result Size
Context Query *context=itemZ*	1600	15998	400 [44134]
Context Query *context=productZ*	1660	14346	400 [971518]
Context Query *context=productU*	152	14024	1 [1561]
Context Query *context=itemU*	66	13,885	1 [540]
Context+Content Query *context=item&content=Lemon Grass*	512	21,000	1 [482]
Context+Content Query *context=category &content=Fruits*	417	33,000	1 [399]
Content Query 1	222000	‡	[8189340]
Content Query 2	1500		[617]

‡ Cannot express such a query in Berkeley XML

Benchmarking with multiple XML documents

50 documents of 1 MB each

Queries	NET MARK	Berkeley XML	Result Size
Context Query *context=itemZ*	2500	Cannot query multiple documents	2,000 [220684]
Context Query *context=productZ*	2610		2,000 [4857608]
Context Query *context=productU*	415		1 [1561]
Context Query *context=itemU*	599		1 [540]
Context+Content Query *context=item&content=Lemon Grass*	602		1 [482]
Context+Content Query *context=category &content=Fruits*	643		1 [399]

100 documents of 1 MB each

Queries	NET MARK	Berkeley XML	Result Size
Context Query *context=itemZ*	5620	Cannot query multiple documents	4,000 [441366]
Context Query *context=productZ*	5630		4,000 [9715214]
Context Query *context=productU*	415		1 [1561]
Context Query *context=itemU*	599		1 [540]
Context+Content Query *context=item&content=Lemon Grass*	602		1 [482]
Context+Content Query *context=category &content=Fruits*	643		1 [399]

4.1.3 Discussion on Query Processing Benchmarking Results

The primary purpose of the above benchmarking exercise was to provide an estimate of query processing times in NETMARK per se. Additionally we also compared it with the Berkeley XML system. The above results show that NETMARK can very efficiently process queries for documents of large sizes, and also large numbers of documents simultaneously. The query processing time also seems to increase with the size of the result set returned. Context and Content queries where the result set size is not large are processed very efficiently, often in the milliseconds range. For smaller result sets, NETMARK significantly outperforms Berkeley XML with most queries being processed 20-40 times faster.

4.2 Netmark Pipeline Benchmarking

We also evaluated the throughput of data insertion into NETMARK. New documents input into the NETMARK system are first converted into XML by the NETMARK converters and then XML documents are loaded into NETMARK node structures in relational tables.

We measured the throughput rate of document conversion into XML. We used two datasets for this benchmark, a set of 50 PDF files (of each approximately the same size) with a total size of 17 MB, and another set of 85 MS Word documents (again

of each approximately the same size) with a total size of 6.5 MB. The document converter was run on a DELL Latitude machine with 1 Intel Pentium 4 CPU of 1.6 GHz and 512 MB of RAM, running Windows XP. The time taken to convert the PDF document set was 84 seconds, thus giving a throughput rate of 1.68 sec per PDF document or 4.94 sec per MB. Conversion of the MS Word document set took a total of 8 seconds thus giving a throughput rate of 0.09 sec per document or 1.23 sec per MB.

5. Related Work

NETMARK is related to several other systems and research efforts in the XML systems area which has seen a flurry of activity in recent years. There is work on XML data management systems[9,11], XML publishing systems[4,5,7], XML query processing aspects[8,10,17,18], integration of XML with Information Retrieval (IR) systems[3], etc.

In this paper we have focused mostly on a new approach in NETMARK to store semi-structured (XML) data in a relational database and the query processing and performance given this storage approach. We should then compare with other approaches to storage and query processing in other XML data management systems. One approach has been to use relational database systems for storing and querying XML data, which is also what NETMARK does, albeit differently in a schema-less way. Such systems essentially work by storing XML documents in underlying relational structures that encode the XML structure through relationships between the tables. XML documents to be stored are "shredded" into rows in these tables. XML queries are converted to SQL queries over the underlying tables, also results (essentially relational tuples) may be converted back to XML before presenting to the user[11,14,16,17]. We have discussed how query processing in NETMARK differs from such systems in the above query processing sub-section and will stress again that the focus in NETMARK is not that of supporting complicated XML queries with path expressions but rather on supporting context and content oriented and hierarchical queries common in enterprise applications.

An alternate direction in XML data management is around building *native* XML stores, i.e., building an XML data management system from scratch. A number of systems such as TIMBER[9], Natix[12] and Tamino[15] fall in this category. The key arguments in favor of native XML systems are that in the approach of translating XML schemas to underlying relational tables, we often end up with a very large number of relational tables in order to effectively capture the rich XML information. Thus even simple XML queries often get translated into expensive sequences of joins over the underlying relational data. Efforts like TIMBER are aimed at developing an efficient *direct* implementation of XML. While we have compared NETMARK with at least one XML over relational system (Berkeley XML), it will be interesting to compare NETMARK with a native XML system from a performance perspective.

6. Conclusions

We have presented NETMARK, a high-throughput and scalable system for managing semi-structured enterprise data. A key distinguishing feature is that of allowing queries on the hierarchical document structure without requiring the imposition of any formal schemas. The schema-less approach and efficient query processing provide a high-throughput system for large real-world applications. We finally demonstrated the efficacy of the NETMARK system and approach through extensive benchmarking results. NETMARK has been and is being used as a building block for several NASA enterprise applications.

7. References

[1] Oracle 9i Database Release 9.0.1 Developer Guide.

[2] A.R.Schmidt, Waas, F., M.L.Ketersen, D.Florescu, Manolescu, I., M.J.Carey and R.Busse, The XML Benchmark Project. CWI, 2001.

[3] Bremer, J. and Gertz, M., XQuery/IR: Integrating XML Document and Data Retrieval.

[4] Carey, M., Xperanto: Middleware for publishing object-relational data as XML documents. *Proceedings of Very Large Databases*, Cairo, Egypt, 2000.

[5] Deutsch, A. and Tannen, V., MARS: A System for Publishing XML from Mixed and Redundant Storage. *20th VLDB Conference*, Berlin, Germany, 2003.

[6] Docushare, http://docushare.xerox.com/ds/.

[7] Fernandez, M., Kadiyska, Y., Suciu, D., Morishima, A. and Tan, W., SilkRoute: A Framework for Publishing Relational Data in XML, *ACM Transactions on Database Systems*, 27 (2002) 438-493.

[8] Guha, S., H.V.Jagadish, N.Koudas, Srivastava, D. and Yu, T., Approximate XML Joins. *ACM SIGMOD*, Madison, WI, 2002.

[9] H.V.Jagadish, Khalifa, S., Chapman, A., Lakshmanan, L., Nierman, A., Paparizos, S., Patel, J., Srivastava, D., Wiwatwattanna, N., Wu, Y. and Yu, C., TIMBER: A Native XML Database, *VLDB Journal*, 11 (2002) 274-291.

[10] Ives, Z., Halevy, A. and Weld, D., An XML query engine for network-bound data, *VLDB Journal*, 11 (2002) 380-402.

[11] J.E.Funderbunk, Kiernan, G., Shanmugasundaram, J., Shekita, E. and Wei, C., XTABLES: Bridging relational technology and XML, *IBM Systems Journal*, 41 (2002) 616-641.

[12] Kanne, C. and Moerkotte, G., Efficient Storage of XML Data (Poster Abstract). *ICDE*, San Diego, CA, 2000, pp. 198.

[13] Maluf, D. and Tran, P., NETMARK: A Schema-Less Extension for Relational Databases for Managing Semi-structured Data Dynamically. *ISMIS*, 2003, pp. 231-241.

[14] Schmidt, A., Efficient Relational Storage and Retrieval of XML Documents. *Workshop on Web and Databases (WebDB)*, Dallas, TX, 2000.

[15] Schoning, H., Tamino: A DBMS designed for XML. *ICDE*, 2001, pp. 149-154.

[16] Shanmugasundaram, J., Relational Databases for Querying XML Documents: Limitations and Opportunities. *VLDB Conference*, Edinburgh, Scotland, 1999.

[17] Shanmugasundaram, J., Shekita, E., Kiernan, J., Krishnamurthy, R., Viglas, S., Naughton, J. and Tatarinov, I., A General Technique for Querying XML Documents using a Relational Database System, *SIGMOD Record*, 30 (2001) 20-26.

[18] Vagena, Z., Moro, M. and Tsotras, V., Twing Query Processing over Graph Structured XML Data. *Workshop on Web and Databases WebDB2004*, Paris, France, 2004.

[19] Xalan, http://xml.apache.org/xalan-j/.

[20] XML, http://www.w3.org/XML/.