August 15, 2008

RT0816
Computer Science 17 pages

Research Report

Surveys on Inverted Index Updating and Semistructured Data Indexing and Aggregation for TAKMI

Kit Chantola

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Report for Research Internship

IBM Tokyo Research Laboratory
July 28 – August 15, 2008

Advised by
Mr. Kohichi Takeda, Manager
Mr. Issei Yoshida, Researcher
Information & Interaction Section

Surveys on Inverted Index Updating and Semistructured Data Indexing and Aggregation for TAKMI

By Chantola KIT
Ph.D. Student (2nd year)
Kitagawa Database Engineering Laboratory
Department of Computer Science
Graduate School of Systems and Information Engineering
University of Tsukuba

Abstract

This report will presents an overview of **TAKMI**® system, **T**ext **A**nalysis and **K**nowledge **MI**ning, created by IBM Tokyo Research Laboratory. Then, it will discuss some issues related to the process of updating new documents and semistructure data (XML) into TAKMI indexing system. By studying on some related research papers, it will propose the possibly efficient solutions for the above mentioned issues.

Acknowledgment

Firstly, I would like to give my great thanks to IBM Tokyo Research laboratory (TRL) which offers me an opportunity as well as financial support for my research internship.

I also would like to acknowledge the contributions of the following individuals and groups in achieving my surveys during my internship in TRL.

Mr. Kohichi Takeda, the manager of Information & Interaction of TRL, who interviewed me and arranged the work plan for my internship in his research section

Mr. Issei Yoshida, a researcher in TRL, Information and Interaction section, who provided great help in introducing me to TAKMI text mining technology as well as his support, comments, and edition on my report, and also his kindness during my internship

IBM employees, especially Ms. Mei Kobayashi, for their gentleness and friendly welcome that I could never expect. I am very delighted with their nice treat and I wish for the opportunity to work with everybody again

Table of Contents

1. Introduction……1

1.1 Text Analysis and Knowledge Mining (TAKMI)……1
1.1.1 Inverted Index of TAKMI……2
1.1.2 Top-k Algorithm of TAKMI……2

1.2 TAKMI Issues…… 3

2. A Survey on Inverted Index Updating…… 4

2.1 Incremental Updates of Inverted Lists for Text Document Retrieval…… 4
2.1.1 Dual-structure Index…… 4
2.1.2 Long List Allocation Policies…… 5

2.2 A Statistics-based Approach to Incrementally Update Inverted Files…… 5

2.3 Fast On-line Index Construction by Geometric Partitioning……..5

2.4 Summary…… 6

2.5 Discussion…….. 6

3. A Survey on Semstructured data (XML) Indexing and Aggregation……..7

3.1 TopX – Efficient and Versatile Top-k Query Processing for Text, Structured and Semistructured Data…… 7

3.2 An Efficient Structure-based Grouping for XML-OLAP…….. 8
3.2.1 Relational Storage for XML Data……..8
3.2.2 TOPOLOGICAL ROLLUP for XML Structure-based Grouping……... 9

3.3 Summary……... 10

3.4 Discussion……... 10

4. Conclusions…….. 11

***References*..... 12**

1. Introduction

With the rapid development of information technology, textual format is a kind of a very flexible way to describe and store various types of information and large amount of data are stored and distributed as text. Therefore, querying the information from the textual data is inevitable and the most important for any organizations such as business, government, and science. So far, there are many researches as well as applications which have been worked with information retrieval in searching for documents of any specific information from textual data, such as web search engines Google, Yahoo! search, MSN search, and IBM text mining, TAKMI. This report will introduce TAKMI, a well-known text mining system produced by IBM Tokyo research laboratory, and discuss the surveys on inverted index updating. In addition to textual data indexing, the report will also investigate the semistructured data (XML) indexing and aggregation for TAKMI since XML has been widely used in recent years.

1.1 Text Analysis and Knowledge Mining (TAKMI)

Text Analysis and Knowledge MIning (TAKMI) [6] is a text mining technology created by IBM researchers in Tokyo Research Laboratory (TRL). TAKMI provides useful knowledge from very large amounts of textual data. Unlike information retrieval and clustering technology, TAKMI finds valuable patterns and rules in text that indicate trends and significant features about specific topics. TAKMI analyzes information in the content of each textual document and extract interesting information that can be provided only by multiple documents viewed as a whole.

For instance, by applying TAKMI to textual databases in PC help centers, they can automatically detect product failures; determine issues that have led to rapid increases in the number of calls and their underlying reasons; and analyze help center productivity and changes in customers' behavior involving a particular product, without reading any of the text.

In order to extract valuable information from textual data, TAKMI provides Top-k functionality for statistical analysis on distribution of terms. Top-k is ubiquitous in the field of Information Retrieval (IR) to define the "Top-k quality" in search results of an IR system. TAKMI calculates the most frequent terms appeared in an arbitrarily-given document subset very fast, so that an end user can repeat trial-and-error analysis by dynamically changing search criteria. Indexing and runtime are the keys for TAKMI interactive text mining. In the next subsection, let us look at a plain description of an inverted index and Top-k algorithms of TAKMI.

1.1.1 Inverted Index of TAKMI

In this subsection, we briefly introduce the mechanism of inverted index structure and how it is used in TAKMI system for keyword aggregation.

Inverted index mainly consists of two components: vocabulary and posting list, stated by Zobel et al. in [10]. A vocabulary stores, for each distinct term *t*, a count of the documents containing the term *t* and a pointer to the beginning of the corresponding posting list. For each term *t* the posting list for *t* is a list of document ids in which each document contains *t*. A posting list may hold extra information other than document id, for example the term frequency in each document or positional information about term occurrences in text. For basic techniques of index construction and implementation issues, please go to [10].

In TAKMI, Top-k keyword aggregation, or simply Top-k, is implemented by using a modified version of inverted index structures for efficient processing of frequent keywords appeared in a dynamically-given document subset.

The most significant difference between inverted index of typical search engines and that of TAKMI is the sort order of terms in vocabulary. In the vocabulary of a standard search engine, terms are sorted in alphabetical order for efficient search for a particular term. On the other hand, TAKMI system has the special vocabulary, called the rank table, in addition to a normal vocabulary as described above. In the rank table all terms are sorted in descending order of frequency in the whole document set. Figure-1 is an example of the rank table and the posting lists employed in TAKMI.

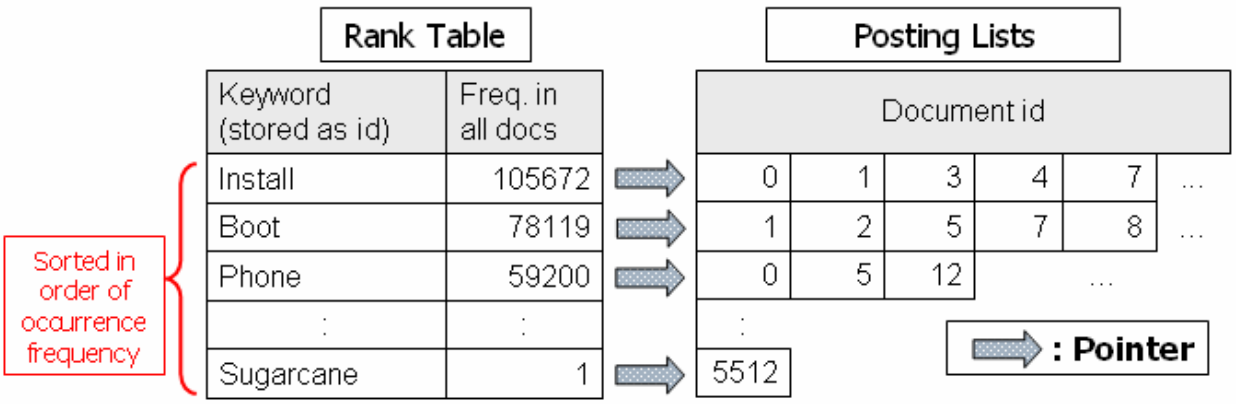

Figure-1: Example of Rank Table and Posting List in TAKMI

1.1.2 Top-k Algorithm of TAKMI

Top-K keyword aggregation is formally defined as follows. Let $D := \{d_1, \ldots, d_n\}$ be a document set to be considered and $Ds \subset D$ be any subset. We can think of Ds as the set of documents retrieved by a search query. Then the Top-K function is defined by:

- **Input:Ds, k (k is an integer)**

- **Output:{$(t_1,f_1),(t_2,f_2),\ldots,(t_k,f_k)$}**

Where t_i is a term and f_i is t_i's frequency in Ds, $f_1 \geq f_2 \geq \ldots \geq f_n$ and t_i's are most frequent terms in Ds.

By sorting terms in descending order of frequency, we can reap the maximum fruit from Threshold Algorithm [12]. Figure-2: shows an outline of TAKMI's Top-k algorithm.

```
Function Top-k:

Input: Ds = {d1, ..., dm} ⊂ D, k
Output:TopK = { (t1, f1), (t2, f2), ..., (tp, fp) | p ≤k, f1≥f2≥...≥fp }

TopK = ∅    // a list
for each (t, f) in the rank table do
   if #TopK < k
      TopK = TopK ∪{ (t, #(Doc(t, D)∩Ds)) }
   else
      (tq, fq) ∈TopK such that fq is minimal in Top-K
      if f < fq
         return TopK    // Exit the loop
      else
         TopK = TopK ∪{ (t, #(Doc(t, D)∩Ds)) } – { (tq, fq) }
end for
return TopK


Function Doc:

Input: D' ⊂D, t:term
Output: Doc(t, D') := {d ∈ D' | d contains t}
```

Figure-2: Top-k algorithm of TAKMI

The key point of the algorithm in Figure-2 is that it processes mainly by sequential access on disks and does not require random access. First we search for the beginning of the rank table and the first posting list, then read each term and its corresponding posting list one-by-one. The algorithm terminates either when the "next" term is less frequent than the least frequent term in the current Top-k list, or when all of the terms in the rank table have been processed.

1.2 TAKMI Issues

Inverted file indexes use a lot of disk space, but searching is fast. Oppositely, according to the characteristic of "disk" addressed in many research papers, such as Lester [3], Shieh [2], Tomasic [1], that updating inverted file is slow since all terms and their pointers are stored contiguously in disk.

Since TAKMI bases on inverted file indexing, the most problem is the costly updating inverted index. For instance, if there are some new documents after TAKMI created its inverted index, a new inverted index including all concepts

from the both existing documents and new documents must be regenerated. Such a kind of updating is not a trivial task as we need to reproduce a large inverted index every time the new documents appear. However, this report is considering ***a survey of how inverted index can be updated efficiently***.

Moreover, while TAKMI is working on textual data analysis, eXtensible Markup Language (XML) is becoming the de facto format for data storage and transmission on the web. Making ***a survey on semistrutured data (XML) indexing and aggregation*** must be a beneficial task for TAKMI as TAKMI does not take the structure of XML data into account for its analysis.

The rest of this report will be divided into two parts of surveys according to the above mentioned issues: 1) A survey on inverted index updating (Section 2), and 2) A survey on semistructured data indexing and aggregation (Section 3). At the end, in Section 4, it will give conclusions.

2. A Survey on Inverted Index Updating

This section reviews three research papers related to inverted index updating in textual data indexing, starting from the paper of primitive technique up to the later papers of more efficient and scalable technique for inverted index updating.

2.1 Incremental Updates of Inverted Lists for Text Document Retrieval

Tomasic et al. [1] addressed the problem of incremental updates of inverted lists using a new structural index. They presented a dual structure index strategy to address the problem of dynamic, time critical text document databases. They also considered on choosing the policies which favor updating time or query time. The following subsections will briefly explain dual-structure index and long list allocation policies.

2.1.1 Dual-Structure Index

Dual-Structure is a new dynamic data structure for inverted lists. The lists are initially stored in a "*short list*" data structure; as they grow, they migrate to a "*long list*" data structure. The objective of dual-structure index is to incrementally update the disk with the in-memory inverted index as efficiently as possible.

Dual-structure index allows us to apply different storage structures to the huge number of infrequent words and to the relatively few frequent words. Through the use of fixed-size buckets, this approach dynamically discovers the frequent words that require their own long list. Updates to the large number of infrequent words are amortized into a relatively small number of disk operations, since the buckets are small enough to fit in memory. In addition, coalescing infrequent words reduces wasted disk space due to allocation of complete disk blocks to very short lists.

2.1.2 Long List Allocation Policies

Once a word has a long list on disk, subsequent in-memory lists for that word will be appended to that long list. Tomasic et al. presented two extreme policies for allocating long list to disk.

They addressed a range of approaches to storing long lists. By varying their defined parameters, they can model schemes that keep the lists sequential and those that break the lists into contiguous chunks

Each policy dictates, among other things, where to find space for a growing list, whether to try to grow a list in place or to migrate all or parts of it, how much free space to leave at the end of a list, and how to partition a list across disks.

2.2 A Statistics-based Approach to Incrementally Update Inverted Files

Shieh et al. [2] proposed a run-time, statistics-based approach to allocate spare space in an inverted file for future updates. The approach determines the size of spare space according to the trade-offs between space efficiency and space utilization. By adaptively balancing the trade-offs, the proposed approach can incrementally update an inverted file as new documents arrive, and in the meantime, the size of unused free space can be well controlled such that the performance of file access would not be affected. The most important key point of the proposed approach is to use simple and recently statistical data to meet the space requirements for an inverted file. This is particularly suitable for in-place updating the indexing structure of all kinds in modern large-scale IR systems, e.g. search engines, or in real-time information systems, e.g. news servers.

2.3 Fast On-Line Index Construction by Geometric Partitioning

While many researches were focusing on off-line merge-based method for inverted index structure, Lester et al. [3] proposed a mechanism for on-line index construction for text databases based on the principle of dividing the index into a small number of partitions of geometrically increasing size. They proposed a scheme that blended the remerge method (their previous proposed method in [8]) and the approach of Tomasic et al., [1]. The key idea was to break the index into a tightly controlled number of partitions. Limiting the number of partitions meant that as the collection grows there must continue to be merging events.

To handle the cost of merging, they limited the capacity of each partition give rise to a natural sequence of hierarchical merges which followed the radix-*r* representation of the number of bufferloads that had been merged to date. Also, *r* was chosen based on the balance of operation of inverted index building cost and access/query cost. Ultimately, in contrast to update mechanism for standard

contiguous representation of inverted indexes, construction costs were significantly reduced, and more scalable.

2.4 Summary

All papers were considering on inverted index structure for efficient updating by proposing various strategic and trade-off techniques for scalable updating. Tomasic et al. [1] seem to be the pioneer of later two papers, Shieh et al. [2] and Lester et al. [3]. While Tomasic et al. proposed a technique of short list (in-memory) and long list (on-disk) for updating long inverted list, Shieh et al. proposed a technique of adding spare space for each inverted list and used statistical method to define the size of spare space. Finally, Lester et al., in contrast to Tomasic and Shieh, proposed an on-line merge-based method which they could update inverted index as soon as the new documents appeared. However, their proposal was the combination of remerging and Tomasic's which index was divided into very limited number of partitions.

2.5 Discussion

According to the three papers, even though the proposed techniques enable them to update inverted index, the possibility is still limited.

Tomasic's as the primitive technique, it still needs a strategy to rebalance the division between short and long lists for any number of incremental updates.

For Shieh et al., although they used statistical method to help them define the size of spare space, this technique still face the problem of spending large disk space whenever the number of documents become large.

For Lester's , the technique seem the best one. Nonetheless, since each index list is in multiple parts, querying is slower than with single-partition list, and it is costly for partitions merging.

Due to the mentioned issues, there is a possibility to improve the index updating technique. That is utilizing the competent strategy of Shieh's for the newest partition of Lester's. Applying Shieh's technique by adding spare space in to the last partition of Lester's, which enable Lester's to increase the speed of updating before it merges the partition with the existing partitions .

In addition to the previous possibility, since the weak point of Lester's technique is the number of partitions, cutting down the partition number will decrease the cost. While Lester's partitioning index by number of documents, partitioning inverted index by both number of documents and terms will reduce the cost of merging as much as the number of documents and terms increase. In this case, the number of partitions may be greater than Lester's partitions, by the way we should play a

strategy of increasing the number of documents in each partition, so that our new technique will lead to less number of partitions than Lester's, especially when the collection of documents becomes very large. Moreover the merging cost will be much improved due to the less number of terms in each partition. Furthermore, in case that a query contains the terms in some of all partitions, we may also get benefit from this technique for improving querying cost by just going to the specific partitions referring to the terms instead of scanning through all partitions,

3. A Survey on Semistructured Data Indexing and Aggregation

This section will address two research papers which the first paper related to semistructured data indexing and the second paper involved to structure-based hierarchical aggregation.

3.1 TopX – Efficient and Versatile Top-k Query Processing for Text, Structured, and Semistructured Data

TopX is an efficient and effective search engine for unstructured, semistructured, and structured data (XML) which was proposed by Theobald et al. [5]. In order to search over all three kinds of data, TopX integrated database and information retrieval techniques by unifying indexing and querying large collection of all kinds of data in a general schema on top of a relational back-end. This report will merely focus on storing structured data (XML) in relational database for IR as this is the main purpose of the survey.

As for the database point of view, TopX provided algorithmic basis for scalable, Top-k-style processing of large amount of data. They stored and queried large document collections with adaptive, disk-oriented cost model, and disk-resident index structures.

For Querying XML data, user would probably start with content only and then refine it to a content and structure query, either manually or with system support.

For Indexing XML data, TopX extended per-term inverted lists to tag-term list of all elements with the tags which hold the search term. They also considered individual element frequencies and element size instead of the document-based counterparts. They stored pre-order, post-order, and level numbers using the XPath accelerator technique. Tag-term lists are stored in an index-only table using the schema TagTermFeature (docid, term, score, maxscore, pre, post, level). They built B^+ index over full range attribute (tag, term, maxscore, docid, score, pre, post, level) and used oracle's index key compression to truncate redundant key prefixes and skip dispensable key prefix at the inner index nodes of the B^+-tree structures. For the detail for TopX index structure, please refer to the paper [5].

3.2 An Efficient Structure-based Grouping for XML-OLAP

Kit et al. [7] proposed a system for XML-OLAP which is constructed on top of relational databases. The system supports both value- and structure-based hierarchies which enable users to make analysis of XML data taking account of XML data features. They used path approach [9] for mapping XML data to relations and then they proposed a new algorithm using a Stack Tree Join algorithm for structure-based, TOPOLOGICAL ROLLUP, grouping operation.

This report will pick up how XML data is stored in relational database and structure-based aggregation (TOPOLOGICAL ROLLUP) for its survey.

3.2.1 Relational Storage for XML Data

They employed the path-approach [9] for mapping XML data to relational tables. In the path-approach, an XML node is basically mapped to a relational tuple of two tables, path table which contains all absolute path expression of all XML nodes, and node table which contains all XML node information. For example, an XML document, “sales.xml”, in Figure-2, contains the books in some area of Japan which each node in the figure is attached with pre-order number, post-order, number, and level. “sales.xml” is extracted into relational tables as in Table-1 and Table-2.

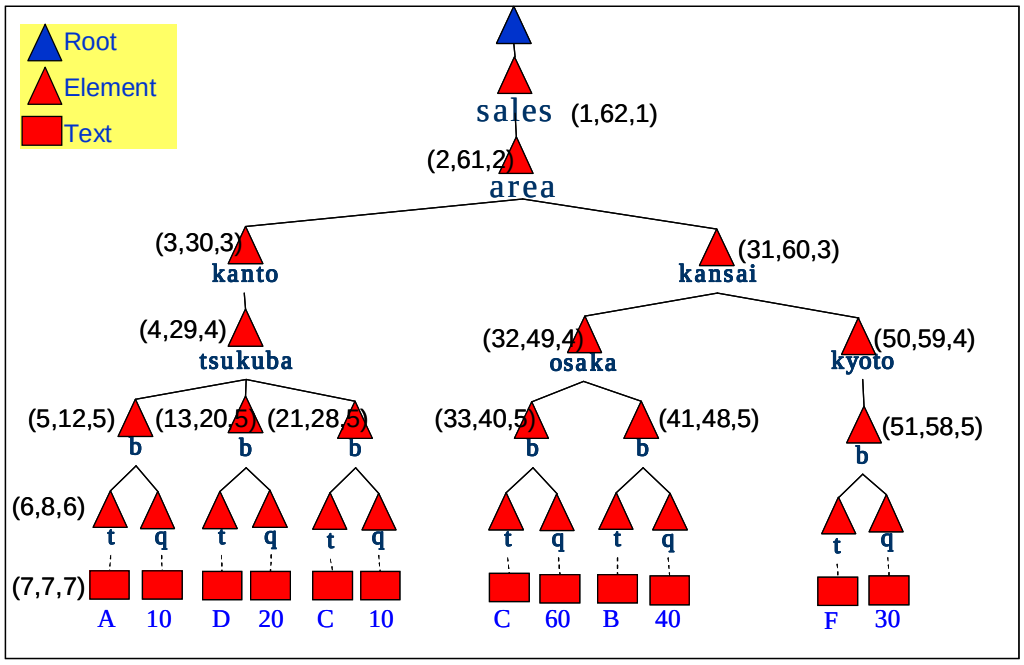

Figure-2: sales.xml, an XML Document Example

pid	pexp
1	/sales
2	/sales/area
3	/sales/area/kanto
4	/sales/area/kanto/tsukuba
5	/sales/area/kanto/tsukuba/b
6	/sales/area/kanto/tsukuba/b/t
7	/sales/area/kanto/tsukuba/b/q
8	/sales/area/kansai
9	/sales/area/kansai/osaka
…	…

Table-1: Path Table of Sales.xml

did	pid	Pre	post	type	value
1	1	1	62	sales	null
1	2	2	61	area	null
1	3	3	30	kanto	null
1	4	4	29	tsukuba	null
1	5	5	12	b	null
1	6	6	8	t	null
1	6	7	7	#TEXT	A
1	7	9	11	q	null
1	7	10	10	#TEXT	10
…	…	…	…	…	…

Table-2: Node Table of Sales.xml

Table-1 shows the path table extracted from “sales.xml”. The attribute *pid* and *pexp* in the path table denote path id to join path table with node table, and the absolute path of XML node. In the node table (Table-2), there are *did*, *pid*, *pre*, *post*, type, and *value*. Attribute *did* denotes the id of the XML document, *pid* is the path id referring to the path expression in the path table, *pre* and *post* are pre-order and post-order number used to identify the node, and *type* denotes the node type, which is either of element name, “#TEXT”, “@attribute”, or “CDATA” depending on the type of the node. The last column is the value of text or attribute node.

3. 2. 2 TOPOLOGICAL ROLLUP for XML Structure-based Aggregation

TOPOLOGICAL ROLLUP is a special syntax for XML data where a number of group-bys are computed according to the hierarchy of an XML data, Table-3 show an example of TOPOLOGICAL ROLLUP grouping the quantity of book by area starting from bottom level (Tsukuba, Osaka, and Kyoto), then upper level (Kanto and Kansai), and top level (Area).

Area	Total Quantity
Tsukuba	40
Osaka	100
Kyoto	30
Kanto	40
Kansai	130
Area	170

Table-3: An Example of TOPOLOGICAL ROLLUP by Area of "sales.xml"

In their previous work, they used UNION ALL, which enabled them to compute set union over different grouping levels, to create subtotals that roll up from the most detailed level to a grand total. Obviously, it is not efficient, in particular, for large XML data. To improve TOPOLOGICAL ROLLUP they used a dedicated algorithm, Structural Tree Join (STJ) [11], to compute structural relationships among sets of XML nodes.

STJ is an algorithm for finding the relationship between sets of XML nodes. This algorithm can help us to find nodes in the same group. STJ required two input lists, ancestor list (*AList*) and descendant list (*DList*). Ancestor list holds ancestor nodes of XML data for grouping and descendant list covers descendant nodes to be grouped by ancestor nodes. Each node of both lists contains *did*, *pre*, *post*, and *value* which can be extracted from node table of relational storage. STJ will compare *pre* and *post* number of each *AList* node with each *DList* node to find the parent-childe/ancestor-descendant relationship. For more detail of STJ, please refer to Al-Khalifa [11].

Kit et al. repeatedly applied STJ ordered by ancestor node algorithm from the bottom (tsukuba, osaka, kyoto) to the top level (area) of XML hierarchical level and efficiently computed multiple groupings which they call TOPOLOGICAL ROLLUP.

3.3 Summary

The TopX engine aims to solve the issue of IR searching over unstructured, structured, and semistructured data, by integrating DB and IR.

While TopX had great work with IR and DB, Kit. et al. [7] were focusing on analyzing XML data by proposing XML-OLAP system which storing XML data to relational database and an effective algorithm, TOPOLOGICAL ROLLUP, for structure-based rollup grouping of XML data.

3.4 Discussion

TopX is mature with textual data Information Retrieval, but the ability to store and aggregate structured data is still limited. Therefore, the report forwardly studied

another paper presented by Kit et al. [7] which they focused on storing XML data into relational database and proposed an effective aggregation function for XML hierarchical structure.

Applying XML-OLAP technique to TopX for storing and aggregating structure data would be worthwhile for TopX in improving IR searching over semistructured data. The possibility is that we can adopt XML-OLAP relational storage technique for TopX posting list.

Then, the algorithm TOPOLOGICAL ROLLUP of XML-OLAP may be beneficial for TopX to improve the cost of computing Top-k ranking by XML hierarchal structure. The reason is TopX simply stores XML data in relational database, any query or aggregation materialize the existing feature of relational database such as using SQL. As mentioned by [7], structure-based grouping of XML data is costly if it bases on SQL. Using UNION ALL in SQL to compute hierarchical grouping, the process multiple groups the measures by all hierarchical levels and, at each level grouping, it scans the whole database and extracts the same elements of measures. Therefore [7] proposed an effective algorithm for XML structure-based grouping using STJ since STJ has proficient joining for structural data, particularly XML data, as mentioned previously in STJ subsection. Further more, the new algorithm extracts only one descendant list for all levels grouping instead of repeatedly extracts descendant lists for each level which is done in SQL.

For TAKMI, TopX are using the same structure of inverted index to TAKMI, so the proposed idea with TopX indexing can be applied to TAKMI for structured and semistructured data retrieval. Moreover, using both applicable techniques of TopX and XML-OLAP aggregation may improve the solely XML-OLAP aggregation performance as advantaged from TopX indexing. Then, TAKMI will be able to retrieve any information from structured and semistructured data for any analysis quickly.

4. Conclusions

This report was divided into two main parts. The first part studied some papers related to inverted index and updating, and discussed on some matters in their proposed techniques and suggested some possible way to enable TAKMI update it’s inverted index efficiently by proposing some possible techniques making use the combination of the two technique of [2] and [3].

Another study (the second part) was about structured data indexing and aggregation. With the first paper of TopX, the study focused mainly on DB for structured data and indexing. For the second paper of XML-OLAP, the study addressed how the authors store structured data (XML) into relational database and their efficient proposed algorithm for structure-based grouping of XML data. Finally, the report proposed a possibility of adopting the relational storage and

aggregation algorithm of XML-OLAP to TopX structured data storage and Top-k ranking computation respectively.

References

[1] A. Tomasic, H. Garcia-Molina, and K. Shoens. Incremental updates of inverted lists for text document retrieval . *In Proc. ACM-SIGMOD Int. Conf. on the Management of Data*, pages 289-300, Minneapolis, Minnesota, May 19944. ACM.

[2] W. Y. Shieh and C. P. Chung. A Statistics-based approach to incrementally update inverted files. *In H. R. Arabnia, editor, Proc. Int. Conf. on Information and Knowledge Engineering*, pages 38-43, Las Vegas, Nevada, June 2003. CSREA Press.

[3] N. Lester, A. Moffat and J. Zobel, Fast on-line index construction by geometric partitioning, *In Proc. ACM Int. Conf. CIKM' 05*, pages 776-783, Bremen, Germany, October 2005.

[4] M. Yoshikawa, T. Amagasa, and T. Shimura. XRel: A pathbased approach to storage and retrieval of XML documents using relational databases. *In ACM Transactions on Internet Technology (TOIT)*, volume1(1), pages 110-141, 2001.

[5] M. Theobald, H. Bast, D. Majumdar, R. Schenkel, and G. WeiKum. TopX – efficient and versatile Top-k query processing for text, structured, and semi-structured data. *In The VLDB Journal*, volume 17(1), pages 81-115, Secaucus, NJ, USA, 2008.

[6] T. Nasukawa and T. Nagano, Text analysis and knowledge mining system, IBM Systems Journal, Vol.40, No.4, 2001, available from http://research.ibm.com/journal/sj/404/nasukawa.html

[7] C. Kit, T. Amagasa, and H. Kitagawa. An Efficient Structure-based Grouping for XML-OLAP. In *The 70th National Workshop of Information Processing Society of Japan technical report*, pp. 5.25-26, Mar., 2008

[8] N. Lester, J. Zobel, and H.E. Williams. In-place versus re-build versus re-merge: Index maintenance strategies for text retrieval systems. In V. Estivill-Castro, editor, *Proc. Australasian Computer Science Conf.*, pages 15-22, January 2004.

[9] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. XRel: a path-based approach to storage and retrieval of XML documents using relational

databases. *In ACM Transactions on Internet Technology (TOIT)*, volume1(1), pages 110-141, 2001.

[10] J. Zobel and A. Moffat. Inverted files for text search engines. *In ACM Computing Surveys (CSUR)*, volum38(2), article 6, 2006.

[11] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J.M. Patel, D. Srivastava, and Y. Wu. Structural joins: a primitive for query pattern matching. *In Proc. of ICDE 2002*, page 141, 2002

[12] F. A. Lotem and M. Naor. Optimal aggregation algorithms for middleware. *In Journal of Computer and System Sciences*, volume 66(4), pages 614-656, 2003.