論 文 内 容 の 要 旨

In recent decades, information is increasing dramatically in size, leading to an urgent demand for high-performance data processing technologies for large volume of data, especially in the field of large XML document processing. For example, Wikipedia [1] provides a dump service [2] in the form of wikitext source and metadata embedded in XML. The sizes of the data was less than one gigabyte before 2006 and the size is over 100 gigabytes nowadays [3]. Some XML documents even reach hundreds of gigabyte. For example, an online database of protein sequence UniProtKB [4] is stored in XML document sized of 358 GB.

Since CPUs were mostly single-core processors at the beginning of this century, XML processing techniques mainly focused on processing XML documents in a sequential way [5, 6]. As the multiple-core processors had gradually became popular then, a lot of approaches on parallel XML processing had been proposed [12, 13]. One common topic of XML processing is the parallelization of XPath [14] queries over XML documents. The basic idea of parallelize is to divide an XML document into multiple parts and process queries separately in parallel. One key problem of the parallelization is how to deal with the hierarchical structure that an XML document has, because this intrinsic characteristic plays a very important role in parallel XML documents processing, and thus forces us to deal with the relationships among these parts after splitting so that we can parallelize the evaluation of XPath queries. For addressing the above difficulty, many approaches had been proposed [18, 19, 20]. Most of these approaches tend to represent XML document in trees and preserve them in memory, then parallelizing XML processing on these trees, which usually utilize multi-thread techniques in a shared memory where all the threads are able to access the shared XML data. However, these studies discuss on the parallelization of XPath queries on trees.

One important topic of XML processing is to exploit database techniques. XML processing in databases has also been widely studied [21, 22, 23]. Common database techniques, such as indexing [28], join algorithms [29], are also valid to be applied to XML processing. Although concurrent transactions are available in modern database engines, to the best of our knowledge, there is no existing work focuses on the parallelization of XPath queries in XML databases. Therefore, to parallelize the evaluation of XPath queries in XML databases, we address the following two challenges.

- Parallelizing Evaluation of a Single XPath Query.
  By dividing or rewriting an XPath queries into multiple subqueries, such as [32, 33], we can convert the evaluation of the original query to the evaluation of these subqueries. However, it is a technical difficulty to figure out how to parallelize the evaluation of a single query by exploiting existing XML database engines, particularly how to achieve good performance gain and scalability.
- Parallelizing Evaluation of queries in distributed-memory environments. When processing XML documents in distributed-memory environments, it is common to partition an XML document into chunks and distribute the processing of chunk to multiple computers. However, how to represent chunks and evaluate queries on them for efficient evaluation, and how to handle the communication among computers are still challenges for efficient XML processing.

For the problems addressed above problems, we proposed two approaches in both shared-memory and distributed-memory environments. We now give a short tour to our study.

The first approach is a development of [33] proposed by Bordawekar et al., which presented approaches to easily exploit esxisting XML processors, such as Xalan, to parallelize XPath queries with no need to modify the processors. Their approach was to partition queries in an ad hoc manner and to merge the results of partitioned queries. Specifically, they proposed three strategies: query partitioning, data partitioning and hybrid partitioning. Query partitioning is to split a

given query into independent queries by using predicates, e.g., from $q_1[q_2$ or $q_3]$ to $q_1[q_2]$ and $q_1[q_3]$, and from q_1/q_2 to $q_1[position() < n]/q_2$ and $q_1[position() > n]/q_2$. Data partitioning is to split a given query into a prefix query and a suffix query, e.g., from q_1/q_2 to prefix q_1 and suffix q_2, and to run the suffix query in parallel on each node of the result of the prefix query. Since hybrid partitioning strategy is a mix of the first two partitioning strategies, we focus on the data partitioning and query partitioning strategies in this thesis. How to merge depends on the partitioning strategies. For query partitioning, it depends on the relationships of subqueries in the predicate. If we split and-/or-predicates, we have to intersect/union results. For data partitioning, if we perform position-based query partitioning and data partitioning, we have only to concatenate results in order. For processing larger XML documents, we extend the study to distributed-memory environment by introducing fragmentation that divides an input XML document into multiple fragment containing information for later querying. We then apply data partitioning on the fragments.

Although Bordawerker et al.'s partitioning approach itself is promising in terms of engineering, their work has already been out of date in this day and age both in the aspect of software environment and hardware environment. It is a technical difficulty to figure out how to parallelize the evaluation of a single query by exploiting existing XML database engines, particularly how to achieve good performance gain and scalability. Furthermore, we now should implement parallelization with serious consideration of underlying XML database engines, in analogy with existing studies on efficient XPath query processing on relational database management systems. There are two questions to be answered: *(1) whether and how can we apply their partitioning strategies to XML database engines? (2) What speedup can we achieve in such case?* In this thesis, we developed two implementations of data-partitioning parallelization and an implementation of query-partitioning parallelization on top of BaseX [32], which is a state-of-the-art XML database engine.

The second approach is based on a novel tree, called partial tree. A partial tree is a tree structure for representing a chunk of an XML document so that XPath queries can be evaluated on partial trees of an XML documents in parallel.
Partial trees can be use in both shared-memory and distributed-memory environments. To understand what a partial tree is, we use the following XML document as an example.

<A><B><C>c1</C><C>c2</C><C>c3</C><A><C>c4</C><B>
</B></A></B><A><B></B></A><B></B></A>

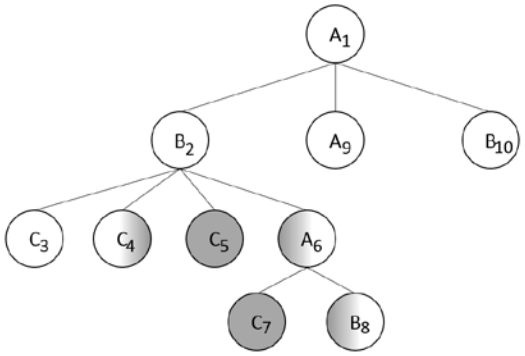

Figure 1.1: An example XML tree

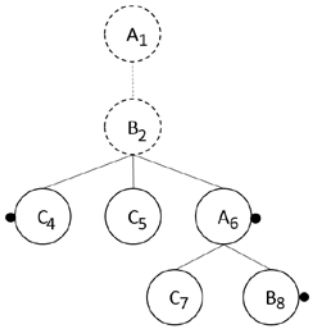

Figure 1.2: Partial tree for the example

We can construct a tree as shown in Figure 1.1 for representing the given XML document. Note that each node in the tree structure is formed by parsing a pair of tags, the start tag and the end tag. The tags in between a pair of tags form nodes as children or descendants of the node formed by the pair of tags. As we know, a node in the tree structure should come from a pair of tags, but not a single tag. Thus, here comes a question: What structure can a single tag represent in the tree? For example, consider the underlined part of the document in Figure 1.1. The corresponding nodes of tags in the underlined part are colored gray in the figure. Note that some tags, such as the first </C> or the last <B>, miss their matching tags. Then, how can we represent these tags when we parse this chunk alone? Besides, $A_1$and B_2 are missing in the chunk. Therefore, how can we apply queries to the gray part in the figure in case we do not know the path from it to the root of the whole tree, i.e. $A_1$and B_2?

To address on the above two problems, we first proposed a novel tree structure, called partial tree. As show in Figure 1.2. We can create a partial tree for the chunk. The partial tree has the information of the missing path, i.e. A1 and B2. This tree is different from ordinary trees because we define some special nodes for partial tree (These special nodes with dots in the feature will be discussed at length in Section 5.1).

Then, by adding the nodes on the path from the root to the current chunk part, we are now able to apply the queries to this partial tree based on the parent-child relationships. We will discuss the query algorithms in Section 5.4. Although partial tree is available in both shared-/distributed- memory environments, it is more specially designed for distributed-memory environments. This is because chunks of an XML documents can distributed to multiple computers and then be parsed into partial trees for further parallel processing.

There are two important contributions for the thesis.
The first contribution involves implementations of [33], and our observations and perspectives from the experiment results. Our implementations are designed for the parallelization of XPath queries on top of BaseX, which is a state-of-the-art XML database engine and XPath/XQuery 3.1 processor. With these implementations, XPath queries can be easily parallelized by simply rewriting XPath queries with XQuery expressions. We conduct experiments to evaluate our implementations and the results showed that these implementations achieved significant speedups. Besides the experiment results, we also present significant observations and perspectives from the experiment results. Based on the results, we extend our study in distributed-memory environments with fragmentation, through which we can distribute size-balanced fragment over a number of computers and apply data partitioning strategy on them.

The second contribution is the design of a novel tree structure, called partial tree, for parallel XML processing. With this tree structure, we can split an XML document into multiple chunks and represent each of the chunks with partial trees. We also design a series of algorithms for evaluating queries over these partial trees. Since the partial trees are created from separated chunks, we can distribute these chunks to computer clusters. In this way, we can run queries on them in distributed-memory environments, utilizing the power of computer clusters. We also presented an efficient implementation of partial tree. Based on indexing techniques, we developed an indexing scheme, called BFS-array index along with grouped index. With this indexing scheme, we can implement partial tree efficiently, in both memory consumption and absolute query performance. The experiment result showed that our BFS-array based implementation can process 100s GB of XML documents with 32 EC2 computers. The execution times were only seconds for most queries in the experiments and the throughput was approximately 1 GB/s.

References

[1] “Wikipedia." https://en.wikipedia.org/wiki/Main Page.
[2] “Wikipedia dump service." https://dumps.wikimedia.org/.
[3] “Size of wikipedia." https://en.wikipedia.org/wiki/Wikipedia:Size of Wikipedia.
[4] The UniProt Consortium, “UniProt: a hub for protein information," Nucleic Acids Research, vol. 43, no.1, pp. 204-212, 2015.
[5] D. B. Skillicorn, “Structured parallel computation in structured documents," Journal of Universal Computer Science, vol. 3, no. 1, pp. 42-68, 1997.
[6] S. Al-Khalifa, H. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu, “Structural joins: A primitive for e_cient xml query pattern matching," in Pro- ceedings of the 12th International Conference on Data Engineering, pp. 141 -152, ACM, 2002.
[14] “XML path lanugage (XPath) 3.1." https://www.w3.org/TR/xpath-31/, 2015.

[18] K. Matsuzaki, Parallel Programming with Tree Skeleton. PhD thesis, University of Tokyo, 2009.
[19] R. Chen, H. Liao, and Z. Wang, “Parallel xpath evaluation based on node relation matrix," Journal of Computational Information Systems, vol. 9, no. 19, pp. 7583-7592, 2013.
[20] W. Hao and K. Matsuzaki, “A partial-tree-based approach for XPath query on 78 large XML trees," Journal of Information Processing, vol. 24, no. 2, pp. 425-438, 2016.
[21] J. Fong, F. Pang, and C. Bloor, “Converting relational database into xml document," in Database and Expert Systems Applications, 2001. Proceedings. 12th International Workshop on, pp. 61-65, IEEE, 2001.
[22] W. Meier, “exist: An open source native xml database," in Net. ObjectDays: In- ternational Conference on Object-Oriented and Internet-Based Technologies, Con- cepts, and Applications for a Networked World, pp. 169-183, Springer, 2002.
[23] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y.Wu, et al., “Timber: A native xml database," The VLDB Journal The International Journal on Very Large Data Bases, vol. 11, no. 4, pp. 274-291, 2002.
[28] E. Popovici, G. Mffenier, and P.-F. Marteau, “Sirius: a lightweight xml indexing 79 BIBLIOGRAPHY and approximate search system at index 2005," in International Workshop of the Initiative for the Evaluation of XML Retrieval, pp. 321{335, Springer, 2005.
[29] W. Liang and H. Yokota, “Lax: An effcient approximate xml join based on clustered leaf nodes for xml data integration," in British National Conference on Databases, pp. 82-97, Springer, 2005.
[32] “BaseX officail documentation”, http://docs.basex.org/.
[33] R. Bordawekar, L. Lim, and O. Shmueli, “Parallelization of XPath queries using multi-core processors: Challenges and experiences”. in Proceedings of the 12th In ternational Conference on Extending Database Technology: Advances in Database Technology (EDBT '09), pp. 180-191, ACM, 2009.
（注）論文要旨：内容は 4,000 字程度とし、紙媒体、Word ファイルで提出
Note: The abstract of the dissertation must be approximately 2000 words and this document must be submitted both on paper and as a Word file.