Parallel Query Processing Algorithms for Semi-structured Data

Wenjun Sun and Kevin Lü

SCISM, South Bank University
103 Borough Road, London SE1 OAA
lukj@sbu.ac.uk

Abstract. Semi-structured data can be described by a labelled directed graph. Query costs on semi-structured data could be very high due to the sequential processing of path matching in a graph shape data structure. In this paper two types of parallel path-based query processing methods are introduced for improving the system performance. The first type includes three parallel version of pointer chasing methods based on the principle of message-driven computation. In the second type of method, the pre-fetch technique is used to achieve a low communication cost and a high degree of parallelisation.

Keywords: Path-based query, Parallel processing, Semi-structured data.

1 Introduction

Queries on semi-structured data depend not only on the contents of the data sets but also on the data schema. A number of methods for query processing and query optimisation have been introduced [1, 2]. These investigations are all based on an assumption that the platform used for query processing is a serial processing environment. The query cost could be very high due to the sequential processing of path matching in a graph shape data structure, in particular, when complex queries (including join operations and multiple path expressions) are involved.

One promising solution is using parallel techniques to improve overall system performance. Parallelism has been employed in both relational and object-oriented databases to improve the system throughput and respond time. To our knowledge, the issues of applying parallel techniques for path expression queries for semi-structured databases have not yet been investigated. Although the data models [3, 4] used to describe semi-structured data are similar to object oriented data models which are graph based models, there are significant differences between them. Because of the features of semi-structured data, the methods used for parallel query processing in traditional database applications will not be suitable, and it poses new challenges to existing data processing technologies.

In this paper, we are going to introduce four newly designed parallel query process algorithms for semi-structured data. The major contribution of this work is that only massage (not data objects, or part of data objects) passing amongst *processing*

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 770-773, 2002.

elements (PEs) during query evaluation. The organisation of the rest of the paper is as follows. Section 2 and 3 describe four parallel query process algorithms. In section 4 presents a summary.

2 Passage Driven Parallel Methods

In this section three new parallel algorithms are introduced for processing semi-structured data. They are the parallel versions of the serial *top-down, bottom-up* and *hybrid methods* [5], namely, Parallel Top-down Query Processing Strategy (PTDQ), Parallel Bottom-up Query Processing Strategy (PBUQ) and Parallel Hybrid Query Processing Strategy (PHQ). These algorithms are designed to explore the possible ways to utilise the resources of a shared-nothing architecture and maximise the degree of parallelisation.

2.1 Parallel Top-Down Query Processing Strategy (PTDQ)

The top-down strategy is a natural way for evaluating a path expression. It follows the path from the root to the target collections using the depth first graph traversal algorithm. It processes the path expression by navigating through object references following the order of the collections in path expressions. If there are multiple references from one object to the objects in the next collection, the navigation follows the depth first order.

2.2 Parallel Bottom-Up Query Processing Strategy (PBUQ)

The parallel bottom-up strategy starts with identifying all objects via the last edge in the input path and check if the objects satisfying the predicate, then traverses backwards through the path, going from children to parents. Similar to the top-down strategy, if it finds that the next object is in another *PE*, it will send an *search instruction* (*SI*) to it. The advantage of this approach is that it starts with objects guaranteed to satisfy at least part of the predicate, and it does not need examine the path any further if the path does not satisfy the predicate.

2.3 Parallel Hybrid Query Processing Strategy (PHQ)

The PHQ strategy start to evaluate a query from both top down and bottom up directions at the same time, and processes from both directions will meet somewhere in the middle. This will increase the degree of paralleisation for query processing. On the direction of top down, the PHQ uses the PTDQ algorithm to create a temporary result which satisfying the predicate so far. Meanwhile, on the other hand, this algorithm traverses up from the bottom vertex using the PBUQ algorithm and reaches the same point as the top-down thread does. A join between the two temporary results yields the complete set of satisfying paths. The detailed algorithms can be found [6].

3 Parallel Partial Match Query Processing Strategy (PPMQ)

The algorithms introduced in Section 3 have provided solutions for processing path expression query in parallel. However, these three algorithms require messages to be passed amongst processing nodes. In addition, a *SI* queue may be built up at a *PE* if it is busy, and it will require further waiting time. To transfer data amongst *PEs* is much more expensive compare other data operations, it should be kept as less as possible.

To overcome these drawbacks, a parallel partial match query processing strategy is introduced. The basic idea of PPMQ is that every *BPE* searches the paths in parallel. When they find an object in a path which is not in their local *PEs*, they will keep records of the identication of the *PE* in which the target object is located rather than send messages to that the descendant *PE*s immediately (where in PTDQ, PBUQ and PHQ, such a message will be sent out). After all possible *partial matched routes* (which will be defined below) have been generated, these *routes* will be sent to one of these *PE*s. In this *PE*, all of *partial matched routes* are merged together and a check is carried out to see if it matches the predicate path expression. During this process, no object (data) or message is required to be transferred between PEs before the finaly merge. This approach could cost less compare to other three approaches, if the operation cost for message passing during the query process in the other three approaches is higher than the cost of final merge in PPMQ, and that in fact it is very much of the case for processing complex queries.

4 Summary

Observing the four algorithms introduced in this paper, it can be found that the key factors that influence the efficiency of these algorithms are the serial fragments and the coefficient of the parallel fragments. The first three methods has two drawbacks: the first drawback is that a large number of communications between PEs may required; the second drawback is when the PEs spinning due to excessive coherence traffic on the communication line causes additional waiting time. As in some cases, PEs must wait for messages from other PEs. This requires synchronisation between PEs and consequently it decreases the degree of parallelisation. The fourth method has been proposed for solving the above drawbacks of first three methods. The main purpose of this strategy is to reduce both communication time and waiting time. However, the price is that it sometimes may need more merging time compare with the other three methods. A linear speed up along with the number of processor could be achieved for the first three algorithms if the waiting time and communication time are considerably low. For the fourth algorithm, a linear speed up could also be possible if there are quite few of number of elements need to be merged. The first three algorithms have a linear speed up apart from the waiting time and communication time. The partial match algorithm also has linear speed up apart from the optimisation time and final merging time.

These algorithms have been coded in C programming language and the system runs under Lunix operating system in a cluster computer system environment. Presently, only a small number of tests have been conducted. In the future, we will

test them under different data sets, query sets with different frequency, to understand more about these algorithms and to find the possible ways to improve them. We also intend to extend the current cost models for these algorithms with additional functions (such as estimate the selectivity, queue waiting time) to develop a set of performance estimation tools for parallel semi-structure data management systems.

References

1. Z.G.Ives, A.Y. Levy, D. S. Weld, Efficient Evaluation of Regular Path Expressions on Streaming XML Data , *UW CS&E Technical Reports by Date* UW-CSE-00-05-02.PS.Z , 2000
2. Curtis E. Dyreson, Michael H. Bohlen, Chriatian S. Jensen, Capturing and Querying Multiple Aspects of Semi-structured Data, *Proc. of the 25th International Conference on Very Large Databases*, Edinburgh, Scotland, 1999.
3. Y.Papakonstantinou, H.Garcia-molina, and Jennifer Widom, Object Exchange Across Heterogeneous Information Sources, in *Proceedings of the Eleventh International Conference on Data Engineering*, PP.251-260, Taipei, Taiwan, March 1995.
4. Document Object Model, *W3C Technical Report*, Nov. 2000, http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113.
5. Jason McHugh, Jennifer Widom, Query Optimization for XML, *Proceedings of the Twenty-Fifth International Conference on Very Large Data Bases*, Edinburgh, Edinburgh, Scotland, 1999.
6. Y. Zhu, W. Sun, K. J. Lü. Parallel Query Processing for Semi-structured Data. Technique report 02-01-KL, SCIMS South Bank University, 2001