2012 International Conference on Information and Knowledge Management (ICIKM 2012)
IPCSIT vol.45 (2012) © (2012) IACSIT Press, Singapore

Efficient GSP Implementation based on XML Databases

Porjet Sansai and Juggapong Natwichai

Department of Computer Engineering
Faculty of Engineering, Chiang Mai University

Abstract. Sequential pattern mining is one of the most important data mining tasks. It can discover important patterns subjected to the minimum support and the time constraints from the transaction data. In this paper, we address the problem of sequential pattern mining on XML databases. More specifically, the GSP algorithm, which is one of the most important sequential pattern mining algorithms, is realized on the XQuery language. The efficient issue of the implementation is addressed based on the XML structure to suit the algorithm. From the experiment results, our proposed approach is much more efficient than the traditional RDBMS implementation.

Keywords: Sequential Pattern, XML databases, Implementation

1. Introduction

As information is the one of the important keys for success in businesses. To get the information from the gathered data, data mining, which is a type of data processing to discover useful information from large databases, has been utilized widely. Subsequently, the information can be utilized to improve the businesses e.g. discovering of customers' purchasing patterns, or determining the appropriate campaign for the customers.

In [1], Srikant and Agrawal proposed GSP algorithm to discover sequential patterns based on the transaction data model. In their model, a database is a collection of transactions. Each transaction is a set of the items $I = \{i_1, i_2, ..., i_m\}$, and is associated with the transaction owner ID and the transaction time. A sequence to be discovered is an ordered list of itemsets. First, the GSP takes each item i in I to form a 1-length candidate sequence $<\{i\}>$, then it scans all 1-sequence candidates that are in the database to get the frequent 1-length sequences, a sequence is frequent if it satisfies the minimum support. Next, the GSP generates the set of 2-length candidate sequence from the frequent 1-length sequences. Then it scans the database to extract all frequent 2-length sequences. The algorithm makes multiple passes over the data until there is no frequent k-length sequence discovered. From the experiments, the GSP is much faster than AprioriAll algorithm. It can also discover all sequential patterns with maximum and/or minimum time gaps and sliding windows.

The GSP algorithm can be considered as a temporal data processing. Since its feature is suitable for implementing on temporal databases. For example, the sliding window, which is one of the GSP-required features, can be implemented on temporal databases directly. With the sliding window, a data-sequence contributes to the support of a sequence by allowing a set of transactions to contain an element of sequence, as long as the difference in transaction-time between the transactions in the set is less than the user-specified windows-size. An example of pattern mining on temporal databases is in [2]. In which, the authors proposed a novel approach to efficiently extract emerging patterns in temporal data using a sliding window coupled with a dual support mechanism. An advantage of the approach is it uses less memory consumption. It also limits I/O and fewer computations by utilizing the previously computed frequent sets. This avoids recalculation of support counts that already exist in between overlapped windows.

Temporal databases were initially introduced in [3] as a type of databases that supports valid-time, transaction-time, or both. In [4], the authors showed a mechanism to query in a temporal database. The temporal relational algebra (TRA) was also introduced. New temporal operators such as "Until" and "Since" or "Coalescing" functions was defined and revised to manipulate temporal data to be able to extract correct information from a temporal database. In [5], the authors focused on directly supporting efficient temporal coalescing queries within existing commercial RDBMS, without any extension to current systems. They proposed two approaches. The first approach takes the advantage of SQL2003's OLAP support. The second approach applies user defined aggregation. Both approaches only need to perform a single scan of the database of the query execution and use minimal joins, which make them possible to provide efficient temporal coalescing. The study demonstrates that current commercial RDBMSs are mature enough to provide efficient complex temporal queries, and they typically also provides a direction for commercial temporal database support within existing RDBMS engine.

For example, a business owner might be interested in the history of product orders bought between 1/1/2012 to 15/1/2012. The SQL statement of such query can be written as follows:

```
SELECT CUSID, BUYDATE, ITEM FROM ORDER, ORDER_DETAIL WHERE ORDER.TID = ORDER_DETAIL.TID AND
(BUYDATE BETWEEN 1/1/2012 AND 15/1/2012)
```

In [5], the authors stated that more complex temporal-queries, e.g. coalescing queries, can be written in SQL. However, the SQL statements can become very complex and often have multiple nested "NOT EXISTS". For example, The SQL statement for the product orders history where the customer id is "2" and between 1/1/2012 to 15/1/2012 is as follows:

```
WITH Temp(ITEMS_BOUGHT, TRANSACTION_TIME AS TSTART, TRANSACTION_TIME AS TEND) AS
(SELECT ITEMS, TSTART, END FROM ITEMS_BOUGHT WHERE CUSTOMER = 2 )
SELECT DISTINCT F.ITEMS_BOUGHT, F. TRANSACTION_TIME AS T_START, F. TRANSACTION_TIME AS
    T_END FROM Temp AS F, Temp AS L
WHERE F. T_START < L. T_END AND F.Items_bought = L.Items_bought AND
    NOT EXISTS (SELECT * FROM Temp AS M WHERE M.Items_bought = F.Item_bought
            AND F. TRANSACTION_TIME < M. TRANSACTION_TIME AND M.T_START < L.T_END AND
            NOT EXISTS (SELECT * FROM Temp AS T1 WHERE T1. Item_bought = F. Item_bought AND
                    T1. T_START < M. T_START AND M.T_START <= T1.T_END))
    AND NOT EXISTS (SELECT * FROM Temp AS T2 WHERE T2. Item_bought = F. Item_bought AND
            ((T2. T_START < F. T_START AND F.T_START <= T2.T_END) OR
            (T2.T_START < L.T_END AND L.T_END < T2.T_END))
```

In [6], the authors showed that the temporal data can be stored and queried efficiently using XML to provide temporally-grouped representations of the database history. In addition, SQL/XML can be used to implement temporal queries expressed in XQuery. From the above query, it can be written in XQuery statement as follows:

```
xquery fn:distinct(for $info in db2-fn:xmlcolumn("ITEMS")/buyinfo/cid[@id = "2"])
for $info2 in db2-fn:xmlcolumn("BOUGHT.ITEMS")/buyinfo/cid/item[.=$info/item]/
    [@T_START < $info/item/@T_END and @T_END <= $info/item/@T_END]
    return < item T_START={$info2/item/@T_START}
    T_END = {$info2/item/@T_END} > { $info2/item } </item>
```

It can be seen that the XQuery language is more expressive than SQL language for this purpose. Since the XQuery language can process semi-structured data efficiently. Also, it is more comprehend than the SQL statements. Furthermore, in the same work, the proposed system demonstrates that the temporal data and temporal queries can be supported efficiently by the proposed features e.g. data clustering and indexing techniques. Subsequently, the approaches are realized into ArchIS (Archival Information System), in which transaction-time capability can be supported to the existing RDBMS and applications efficiently.

In this paper, we propose an implementation of GSP algorithm on XML Databases. The structure of the XML data suited for the algorithm is considered, since such structure can affect the efficiency of the algorithm. Also, the GSP algorithm is implemented by the XQuery language. The proposed work will be evaluated by the experiments. In which, our proposed work will be compared with the algorithm implemented on traditional RDBMS.

2. XML Approaches for GSP Implementation

2.1. Transaction Data in XML Structure

First, the structure of the XML data is considered. As mentioned in Section 1, each transaction contains transaction owner, transaction time, and the itemsets. We propose to store the transaction as the structure in Fig. 1a). It can be seen that the itemsets by the same transaction owner are stored in a single XML tag (own). And, the transaction owner identifier is presented as an attribute (id). Within an owner tag, the transaction time of each item is identified by the "time" attribute. For example, it can be seen that transaction owner with id 1 has item 1 and 5 at the time stamp 6. This approach is based on the fact that XML parsers, in general, have to split the opening and closing tag when the enclosing data are to be read. Thus it can be efficient since the transaction owner identifier, which might be scanned multiple times, can be read without re-traversal into the tree structure of the XML data [7]. This is also the case for the transaction time which is presented as an attribute.

```
<transactions>
 <own id="1">
  <item time="6">1</item>
  <item time="6">5</item>
  <item time="7">6</item>
  <item date="7">35</item>
  …
 </own>
  ...
 <own id="100">
  ...
 </own>
</transactions>
```

a) XML Structure

```
Let Ck denote the set of candidate k-itemsets.
Let Owner denote all transaction owners.
for each (c ∈ Ck){
   for each owner in Owner
      Parse XML element by the white spaces.
      XQuery1: Determine the first time stamp of supporting transactions.
      for each time stamp from XQuery1
        for each XML element
           XQuery2: Determine the element with minimum time stamp.
           if the element satisfies the time constraints
                Increase the support count for the candidate.
```

b) GSP *k*-length candidate verification

Fig. 1: XML Structure and Algorithm.

2.2. GSP Algorithm Implementation

For the algorithm, it is almost a straight forward implementation besides some part of it, which needs to query from the XML data. We present only the *k*-length candidate verification part, since the rest of the algorithm is exactly the same as the GSP algorithm. The algorithm is presented in Fig. 1b). It can be seen that the algorithm composes of the candidate verification loop (C_k), the transaction owner loop (*Own*), and time constraints loop. In the transaction owner loop, the time stamp, which such owner starts to support the candidate, needs to be determined from the XML structure. It is executed by XQuery 1 as shown in Fig. 2a).

And, the minimum time stamp of the supporting transactions within the time constraints is determined by XQuery 2 as shown in Fig. 2b).

```
xquery fn:distinct-values(for $info in db2-fn:
xmlcolumn("DBT2000XMLV6.INFOGSP")/
transactions/own[@id="1"]
return $info/item[.="20"]/@time)
```

a) XQuery 1

```
xquery fn:min(for $info in db2-fn:
xmlcolumn("DBT2000XMLV6.INFOGSP")/
transactions/own[@id = own]
return $info/item[.=\"" + patternArr[p] + "\" and @time >= "
+ window_time + " and
@time <= " + (window_time+ max_gap) + "]/@time)
```

b) XQuery 2

Fig. 2: XQueries.

3. Experiment Results

In this section, the efficiency of the proposed GSP implementation on XML database is evaluated. The experiment was conducted on an Intel Core i5 2.4 GHz PC running Windows7 OS, with 4GB memory and 500 GB SATA hard drive. The GSP algorithm is implemented on two different database structures: XML data, and Relational data. IBM Quest Market-Basket Synthetic Data Generator was used to generate datasets with 20000 transactions for our experiments. IBM DB2 Express-C 9.7.5 is selected as the database engine since it supports both traditional RDBMS and XML. The datasets were stored in the two mentioned database structures. In each experiment, the execution time represented the efficiency form multiple runs are reported.

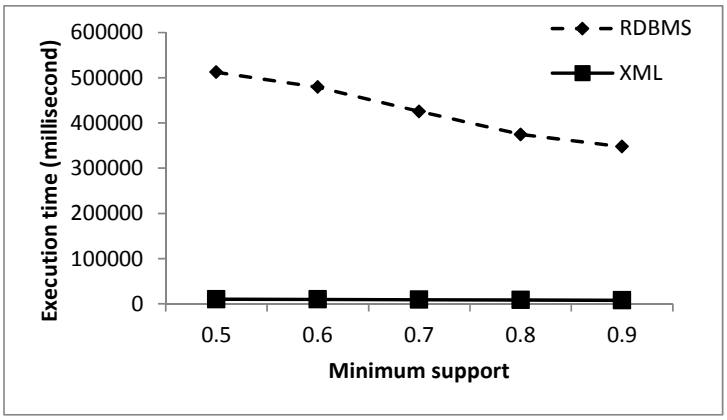


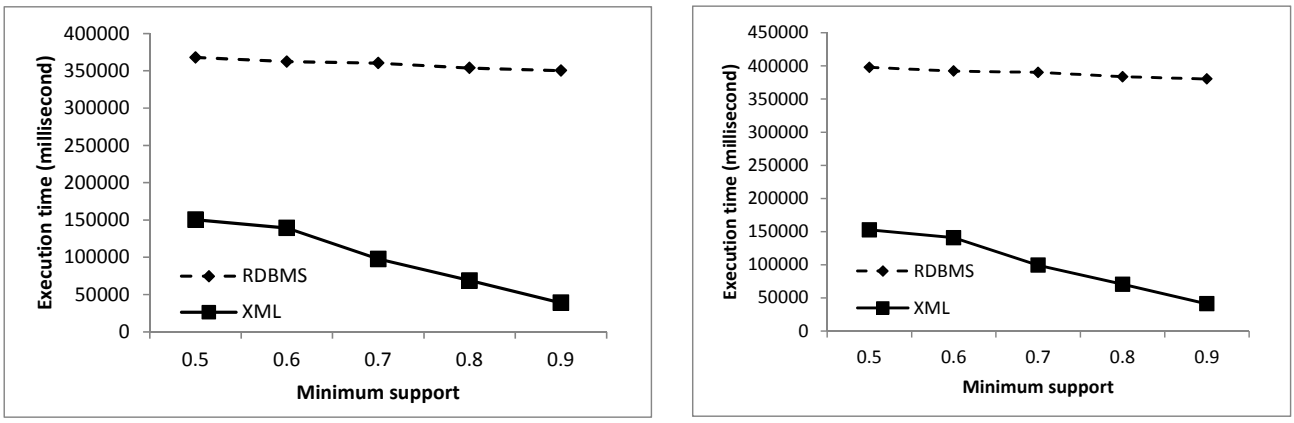
Fig. 3: Execution time without temporal constraints.

Fig. 3 shows the experiment result represented the execution time of sequential pattern discovery without any temporal constraints. On the x-axis, the minimum support is varied from 0.5 to 0.9. On the y-axis, the execution time is reported.

From the result, the execution time of the GSP algorithm implemented on XML database is much lower, approximately 30% less than the RDBMS implementation. Also, the difference is larger when the minimum support decreases. The reason behind this is the RDBMS has to scan every transaction for determining whether an item exists in the transaction. While, in the XML database, such unnecessary multiple scan can be avoided by the proposed approach.

Subsequently, the proposed work was evaluated for the GSP features, i.e. sliding window and the maximum time gap. In Fig. 4a), the sliding window is investigated; it is fixed at 5 time-point. And, in Fig 4b), the execution time when the maximum gap is fixed at 5 time-point is investigated. From Fig. 4, it can be seen that when the time constraints is applied, the XML approach is less efficient. This is because of the time constraints enforce the algorithm to determine the minimum time stamp of the supporting transactions in XQuery 2, which requires more computation expense. However, when the support is increased, the difference between the two approaches is larger, since the XQuery 2 can eliminate the candidates which

satisfy the time constraints with less cost. In summary, it is clear that the implementation on XML structure is much more efficient than the RDBMS implementation.



a) Sliding window effect

b) Maximum gap effect

Fig.4: Execution time with time constraints feature.

4. Conclusion

In this paper, we have proposed an approach to implement the GSP algorithm on XML databases. The XML structure, which can provide a high efficiency, have been proposed as well as the corresponding algorithm based on XQuery language. The proposed approach has been validated by the experiments. The results have shown that the XML-based implementation is much more efficient than the RDBMS approach. Also, it has been found that the time constraints degrade the efficiency of the proposed approach. However, the XML approach can still outperform the traditional RDBMS approach.

5. References

[1] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvement. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, 1996, Springer-Verlag, London, UK, pp. 3-17.

[2] M. Sulaiman Khan, F. Coenen, D. Reid, R. Patel and L. Archer. A Sliding Windows based Dual Support Framework for Discovering Emerging Trends from Temporal Data. *Knowledge-Based Systems*, 2010, 23(4), pp. 316-322.

[3] C. Jensen. *Temporal Database Management*, Ph.D. Thesis, 2000. Department of Computer Science, Aalborg University.

[4] J. Patel. *Temporal Database System*, Master Thesis, 2003. Department of Computing, Imperial College, University of London.

[5] X. Zhou, F. Wang, and C. Zaniolo. Efficient Temporal Coalescing Query Support in Relational Database Systems. In *Proceedings of the 17th international conference on Database and Expert Systems Applications (DEXA'06)*, 2006, pp. 676-686.

[6] F. Wang, C. Zaniolo, and X. Zhou. ArchIS: an XML-based approach to transaction-time temporal database systems. *The VLDB Journal*, 2008, 17(6), pp. 1445-1463.

[7] F. Özcan, D. Chamberlin, K. Kulkarni, and J.-E. Michels. Integration of SQL and XQuery in IBM DB2. *IBM System Journal*, 2006, 45(2), pp. 245-270.