Enforcing Scalability: An Efficient Approach for Distributed Database Partial Replication

Tin Myint Naing, and Aung Win

Abstract—Distributed applications and its relying environments have been growing rapidly along with the advances of technology day by day. Distributed transactions are rather expensive in OLTP setting and it is one of the issues in distributed framework that have to try to minimize these loading as possible. The standard way to scale in distributed OLTP is to horizontally partitioning data across several nodes. This makes the transaction to be executed at one node and could avoid the distribution overheads and allow the system to scale up by adding nodes. In this paper, a new scheme is shown up to try to reduce the substantial amount of distributed transactions allowing a workload to be efficiently scaled across several machines. Firstly, the query traces have to be monitored and logged to perform necessary analysis to catch up the distributed transactions. Second, transactions are converted into a hyper-graph representation based on the result of analysis. Then created graph is correctly partitioned into several clusters by multilevel k-way hyper-graph partitioning method for load balancing in distributed environment. A mapping table is used to enhance the performance penalty of the system as well. Finally, the resultant partitioning is verified by using random forest in order to produce k-balanced partitions for underlying environment. The tested result shows that the proposed approach is well done in distributed environment testing and can handle the increasing workload with multiple database servers.

Keywords—Distributed Transaction, OLTP (Online Transaction Processing), Partitioning, Random Forest.

I. INTRODUCTION

In a distributed environment, data is replicated in order to achieve shorter response times, higher throughput and increased availability and reliability in case of failures [4]. The main idea in replication is to keep several copies or replicas of the same resources at various different servers [6]. Moreover, scalability, reliability and availability are related important issues in implementing distributed applications. It is important to handle the overhead problem when trying to get better performance for one factor. Otherwise, it would only produce the performance penalty for the system. Workload balancing in distributed application environment is also crucial and an efficient partitioning scheme can help to accomplish the desired goals. By placing partitions on different nodes, it is often possible to accomplish nearly linear speedup, especially for analytical queries where each node can scan its partitions in parallel.

Distribution reduces the network costs for query access, and it improves application availability and consistency [6]. There are two main replication strategies for distributed database. Synchronous replication propagates any changes to the data immediately to all existing copies. Moreover, the changes are propagated within the scope of the transaction making the changes. The ACID properties apply to all copy updates. Asynchronous replication first executes the updating transaction on the local copy. Then the changes are propagated to all other copies. While the propagation takes place, the copies are inconsistent (they have different values). The time the copies are inconsistent is an adjustable parameter which is application dependent [7].

Distributed transactions are rather expensive. And so obtaining the optimal partition is critical to gain good performance from a distributed OLTP database [3]. So, graph partitioning algorithm plays a vital role for this proposed distributed framework applications. There exist two general classes of methods for the graph partitioning problem, exact methods which work out the optimal partitions, and heuristic methods which try to quickly compute an approximate solution. Heuristic methods include spectral methods, geometric methods, multilevel schemes, optimization-based methods, and methods that employ randomization techniques such as genetic algorithms [9]. Some researches [1 8 2] have been carried out for automatic partitioning schemes and some approaches are used widely in particle applications. Among them, round-robin, range or hash partitioning are effective for analytical queries that can scan large datasets. But none of these approaches is ideal for workload of small transactions that touch only a few records [5]. The partitioning problem becomes harder and difficult when transactions are associated with multiple nodes with multiple tables. In this work-in-progress paper, a new efficient scheme for distributed database is introduced to gain the optimal performance measures in distributed application environment.

In this framework, a graph representation model is used to represent the distributed transactions which access multiple database servers. The tuples or records of a table stand for the nodes or vertex of a graph and transaction that accesses the nodes become the hyper-edge of that vertex. It is reasonable to represent transactions as hyper-edges in graph and that edges

Tin Myint Naing is with the University of Technology (Yatanaporn Cyber City), Pyin Oo Lwin, Myanmar (corresponding author’s phone: +95 9420703151 ; e-mail: tinmyintnaing08@gmail.com).

Dr.Aung Win, is with the Principle of University of Technology (Yatanaporn Cyber City), Pyin Oo Lwin, Myanmar (corresponding author’s phone: +95 900000 ; e-mail: yeyint2@gmail.com).
will be cutting to get balanced partitions that minimizes the weight of cut edges which approximately minimizes the number of multi-sited transactions. The partitioning is done using various criterions. Generally, the criterion is to minimize the number of hyper-edges that have nodes in different partitions or alternatively the minimization of pin count is used [10].

II. OVERVIEW

The overall procedure of proposed paradigm is shown in Fig. 1. The primary source of input is database workload (SQL traces), and desired partitions. The output is partitioning strategy for workload balancing that minimizes the undesirable distributed transactions. The basic steps of approach are as follows:

Obtaining and analysis of query traces: Query traces that are coming from the clients are logged in system by logger automatically. Read and write sets of transactions of sql traces are computed and made analysis to get pre data for graph representation.

Creating graph representation: A graph representation of the database workload is created based on the observation of preceding step. Tuples of database represent vertex in graph and edges are the usage of tuples within the transactions. An edge connects two tuples if they are in the same transaction.

Partitioning the graph: k-way hyper-graph partitioning algorithm is used to get balanced minimum-cut partitions. Each tuple is assigned to its corresponding partition and each partition is assigned to one physical node, database server.

Refinement partitions: Random forest approach is used to extract a set of rules that compactly represent per-tuple partitioning. The rules used are predicated on the values of the frequent attribute set that map to the partition number. Frequent attribute set can be obtained from sql traces by analyzing the where clause for each tables.

III. DATA ACQUISITION

Obtaining and analysis of query traces: It is the first step and primary process for the system. Generally, SQL query traces are taken from log of database server(e.g, the MySQL general-log). An alternative and effective way to obtain the SQL traces is writing a logger tool to record all incoming SQL in a specified log file in the system. It is more effective and easy to get required log file within a least time when multiple database servers are used. First, SQL statement from the traces are rewritten into SELECT statement that retrieve the identifiers of each tuple accessed. When these SELECT statements are executed, a list of (id,transaction) pairs is issued and it is used to create graph representation.

Some heuristic procedures have to be carried out in order to reduce the graph size and to support in partitioning process.

The first one is transaction level sampling which can limit the size of workload trace represented in the graph, reducing the number of edges. The next one is tuple-level sampling which can also reduce the number of nodes(tuples) in the graph. Lastly, filtering is done to discard occasional statements that scan large portion of a table, as they could produce many edges carrying little information and filtering is needed to be done to remove the tuples that are accessed rarely from the graph.

IV. GRAPH PROCESSING

A. Graph Representation

While the proposed framework works well with multiple tables database schemas, only a very simple example is introduced for illustration. Suppose in a TPC-C database, an stock table includes seven tuples and a workload of four transactions as shown in Fig.2. Each tuple is represented as a node in the graph and edges are connected between two tuples that are used in the same transaction. An edge connects two tuples if they are in the same transaction.

<table>
<thead>
<tr>
<th>s_w_id</th>
<th>s_i_id</th>
<th>s_qty</th>
<th>s_order_cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>90</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>90</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>95</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>60</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>120</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>125</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>90</td>
<td>50</td>
</tr>
</tbody>
</table>

By running the four transactions, the data changes will be seen in Table I. Transactions are shown in above figures in order and the affected nodes and edges can be seen in fig. Node 22, node 23 and node 24 are accessed from two different transactions.
The above graph representation is drawn from the result of sample four transactions. The first one is the updating the stock table by ordering 15 units for \( s_i_id = 3 \). It includes a select and two updates for that ordering. The second transaction is also ordering 50 units for \( s_i_id = 4 \). It consists of a select and two updates for \( s_i_id = 4 \) as well. In transaction 3, the ordering is conducted for that tuples that are greater than 30 units in \( s_qty \) and \( s_i_id \) is less than or equal to 2. The transaction 5 is also for ordering the tuples which are less than or equal to 100 units and \( s_w_id \) is equal to 2. The based values for all tuples is 100 units in \( s_qty \) and 0 in \( s_order_cnt \). The data shown in Table 1 is final result of all updates of four transactions.

### B. Graph Partitioning

A graph partitioning algorithm, k-way graph partitioning for hyper graph, is utilized in order to get a balanced minimum-cut partitioning of the graph into k partitions. Each tuple is assigned to its corresponding partition and each partition is assigned to one physical node, database server.

Graph partitioning splits the graph into k non overlapping partitions such that the overall cost of the cut edges is minimized while keeping the weight of partitions within a constant factor of perfect balance. So this graph operation approximately minimizes the number of distributed transactions while balancing the load evenly across nodes.

K-way graph partitioning is a known to be an NP-complete problem. However, given its fundamental role in VLSI and CAD it has been the subject of substantial research and development efforts over the past four decades[11 12 13], resulting sophisticated heuristics and well optimized, freely available software libraries.

Most graph partitioning algorithms exploit multilevel coarsening techniques and provide distributed parallel implementations to deal with extremely large graphs [3]. In this approach, METIS [14] is used to partition graph. The result of graph partition phase is a fine-grain mapping between individual nodes and partition labels. The result of the graph partitioning for two partitions is shown in Fig 3.

### V. Refinement

The refinement phase try to find the compact model that captures the ( tuple, partition ) mappings produced by the partitioning phase. We utilize random forest machine learning classifier to conduct this task. The collection of (value,label) pairs are taken as main input. In our approach the values are database tuples and the labels are the partition assigned by the graph partitioning algorithm.

To implement this phase by using Weka, it is necessary to create a training set. Data acquisition phase extract queries and compute necessary analysis from the workload trace. Some analysis data such as node weight and edge weight are used to combine with the resultant output of partitioning trace and are used as core input for our approach. Attribute selection is also needed to conduct to find the candidate attributes for each table. As an example, for the TPC-C stock table we obtain two frequently used attributes (\( s_i_id, s_w_id \)), representing item ids and warehouse ids. The candidate attributes are fed into Weka’s correlation-based feature selection to select a set of attributes that are correlated with the partition label. For TPC-C, this step discards the \( s_i_id \) attribute, leaving \( s_w_id \) as the sole attribute for classification.

We build a classifier by using random forest in Weka. Partition labels, number of distributed transactions, number of affected node weight in a transaction are candidate attributes and partition labels are the classification attribute we wish to learn. As an example, for TPC-C with two warehouses divided into two partitions we obtain the following rules for the stock table:

\[
\begin{align*}
\text{s_w_id} \leq 1 & : \text{partition: 1 (pred. error: 1.49\%)} \\
\text{s_w_id} > 1 & : \text{partition: 2 (pred. error: 0.86\%)}
\end{align*}
\]

The output of the classifier for the item table is:

\[
\text{<empty> : partition: 0 (pred. error: 24.8\%)}
\]

This indicates that all tuples in the table are replicated. The high prediction error in this example is an artifact of the sampling, since some tuples in the item table end up being accessed only by few transactions, thus providing no evidence that replication is needed.
VI. ROLE OF LOOKUP TABLE AND QUERY ROUTER

The basic operation of lookup tables and query router is that when a router receives a query from the application, it must determine which backends store the data that is referenced. In general, the queries must broadcast to all the backends machine in order to generate the distributed queries. Thus, the use of lookup tables and query router component can reduce the processing time significantly. Lookup tables are stored in memory, and consulted to determine which backends should run each query. Lookup tables map from a key to a set of partition ids that store the corresponding tuples.

This allows the administrator to partition tuples in an arbitrary (fine-grained) way. Furthermore, lookup tables can be used as partition indexes, since they specify where to find a tuple with a given key value, even if the table is not partitioned according to that key [15].

For queries referencing a column that uses a lookup table, the router consults its local copy of the lookup table and determines where to send the query. In the case of simple equality predicates, the query can simply be passed through to a single backend. If multiple backends are referenced (e.g., via an IN clause) then the query is rewritten and a separate query is sent to each backend. The results must then be merged in an appropriate way (e.g., via unions, sorts, or aggregates) before being returned to the client. More complex queries may require multiple rounds of sub-queries.

VII. EXPERIMENTAL EVALUATIONS

Various trials have been made with standard transaction processing benchmark, TPC-C, in order to gain the nearest approximation of performance analysis of proposed scheme. The structure of TPC-C database is listed in table II. The workload is defined by a mix of 5 transactions selected at random according to the balance of the percentage value stated in tpc-c publication. The percentage of transaction types are stated in table III. The followings are made in very low end PC of core i5 processor with 4 GB RAM and 1 GB graphic card.

<table>
<thead>
<tr>
<th>Table</th>
<th>Number of Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouse</td>
<td>n(specified in a measurement)</td>
</tr>
<tr>
<td>Item</td>
<td>100,000</td>
</tr>
<tr>
<td>Stock</td>
<td>n * 100,000</td>
</tr>
<tr>
<td>District</td>
<td>n * 10</td>
</tr>
<tr>
<td>Customer</td>
<td>3000 per district</td>
</tr>
<tr>
<td>Order</td>
<td>Number of customers(initial)</td>
</tr>
<tr>
<td>New order</td>
<td>30 % of orders(initial value)</td>
</tr>
<tr>
<td>Order line</td>
<td>approx. 10 per order</td>
</tr>
<tr>
<td>History</td>
<td>Number of customers(initial)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name of transaction</th>
<th>Share of all transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>New order</td>
<td>≤ 45%</td>
</tr>
<tr>
<td>Payment</td>
<td>≥ 43%</td>
</tr>
<tr>
<td>Order status</td>
<td>≥ 4%</td>
</tr>
<tr>
<td>Delivery</td>
<td>≥ 4% (batch transaction)</td>
</tr>
<tr>
<td>Stock level</td>
<td>≥ 4%</td>
</tr>
</tbody>
</table>

Fig. 4 Throughput comparison of proposed approach and asynchronous primary copy full replication scheme

Fig. 5 Transaction Response Time for Four Servers

The throughput of TPC-C is a direct result of the level of activity at the terminals (clients). Each warehouse has its specified terminals and all five transactions can be executed at
each terminal. The random generation is emulated to get an equal number of new order and payment transactions and to produce one delivery transaction, one order-status transaction, and one stock-level transaction for every ten new-order transactions. The throughput comparison of partially replicated scheme and asynchronous update everywhere replication scheme is depicted in Fig. 4. The result shows that the partial replicated scheme is slightly better than the fully replicated scheme.

Transaction response time (TRT) is also stated in Fig. 5. The experimental result shows that the response time is not so difference in server 1 or 2, but it is significantly difference in more servers as a result of full replicas. However, partial scheme is slighter shorter in total turnaround time for such a transaction of updates. For partial scheme, the partitioning time for TPC-C workload is shown in Fig. 6. It is not a burden for the proposed scheme as it is stable for required servers count.

VIII. CONCLUSION

A new framework for distributed database partial replication is proposed in this paper. Data acquisition phase is rather important in this proposed approach. Because it is directly affect on the graph representation and so is the partitioning phase. Moreover, some analysis data are also used in refinement. The only drawback of proposed approach is that cross-partition distributed queries can produce distributed overhead. However it can reduce such overhead by repartitioning the database and reallocating the data to several servers. The graph partition based partial scheme is more advantage in scalability and throughput of the system. The proposed scheme can handle the increasing workload and the performance is not so declined when more servers are added.

REFERENCES


Tin Myint Naing was born in 1975 and he finished his high school in 1993 in Yangon. After he passed the grade 10, he joined Institute of Computer Science and Technology, Yangon. Now that institute is renamed the University of computer Studies, Yangon. He got his first degree, B.C.Sc from that University at 2000, DEC. He got the master of computer science degree (M.C.Sc) at 2003, OCT from that university. Now he is doing research in University of Technology, Yatanarpon Cyber City for his Doctor of Philosophy.