Virtual Rule Partitioning Method for Maintaining Database Integrity

Feras Hanandeh, Hamidah Ibrahim, Ali Mamat, and Rozita Johari
Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia

Abstract: *The variant execution time of the update operation repair rules in a parallel and distributed environment is highly affected by the way the rules and tables settled in a database in according to whether they were partitioned or not. The well-known partitioning methods in the database were succeeded to reduce the response time of update operations since they expeditiously execute the update operation affecting different sites of partitions. These methods are mortgaged by the determination of the type of partitioning during the design state of the system fixing all sites of the partitioned tables and distributed rules to different nodes. Although distribution and partitioning have these merits, it still face some restrictions since it may be time consuming for the distributed system to locate the proper rule and the partition of data, which fulfill the requirement of repair update operation. This paper proposes virtual rule partition method. This method has more advantages over the classical methods because it allows us to reduce the total cost or the total response time consumed by repair update operations using horizontal partitioned tables.*

Keywords: *Active database systems, database partitioning, symantec integrity maintenance.*

Received April 2, 2003; accepted August 10, 2003

1. Introduction

The reliability of information systems is a major concern for today's society and enterprises. The correctness or maintaining database integrity of databases is one of the main reliability issues. Consequently procedures asserting correct databases are a chief focus of research. Today the prime obstacles applying these procedures are their high computational costs. Integrity maintenance is considered one of the major application fields of rule triggering systems. In the case of a given integrity constraint being violated by a database transition these systems trigger update operation (action) to maintain database integrity. The variant execution time of an action (update operation) in a parallel and distributed environment is highly affected by the way the rules and tables settled in a database. Parallel execution of the update operations implies the distribution of the execution of these actions into different partitions. Rule triggering systems will then initiate for each location specific sub process through a request from the demander (the generator of the update operation) towards the subscribers (the processing units) and vice versa. The total time by a given update operation to answer a given query applied to a parallel database system is greatly affected by some events [9]. One of the events is the physical location of tuples that could satisfy the conditions emitted by the query. Another event is the number of participants or quorum size.

The main idea of distributed database is to partition all database entities like tables and rules having certain homogeneity criteria and locate each coherent group of information in different sites. This will simplify dealing with its subgroups containing homogeneity data avoiding dealing with the whole data, which may contain great amount of data.

This paper is organized as follows. In section 2, we provide some related work on the partitioning and fragmentation. In section 3, we define some terminologies related to the time required to satisfy a given action, the term Update Operation Repair Rule (*UORRs*) is used in our work. Section 4 discusses the physical partitioning with examples. Section 5 presents our proposed virtual rule partitioning method. Section 6 presents the results of our proposed method. Finally, in section 7 the conclusion of this work is addressed.

2. Related Work

Semantic Integrity Maintenance Systems and database partitioning are a very closed area of research, since many researchers have proposed building Integrity Maintenance Systems exploiting the advantage of database partitioning to ease the process by reducing both time and cost. Partitioning strategies should be included in the definition of tables and need algorithms to locate such partitions [5, 7]. On the other hand Semantic Integrity Constraints declarations specify additional properties or relationships of attributes and relations, which cannot be included in the definition of tables [2].

In parallel database systems the need for partitioning the relations become so essential since a

great amount of data need to be accessed during the execution of a query. Different processors access different partitions of the relation at the same time. Various forms of data partitioning are described in the literature. Hash-based horizontal fragmentation [1], since this form enables parallel query. Logical notation [5] specifies the way in which a database is partitioned. This notation is first used to specify how relations are partitioned in one or more dimensions. [5] introduced the concept of region. A region is a collection of partitions taken from database relations. The partitions that are gathered into a region will contain data, which is semantically interrelated in some way. When complex partitioning strategies are applied to a relation, tuples may be required to migrate between partitions even if they remain unchanged by an update. As partitioning strategies become more complex, the costs of managing tuple migration rise.

In order to achieve database fragments in distributed database, researchers have proposed a number of different and diverse mechanisms [4, 5, 8, 9] the most successful being the horizontal, vertical, and mixed fragmentation methods. These fragmentation methods can be used in parallel database systems [5] where no centralization for the relations is expected. Different processors access more than one fragment the same time.

3. Preliminaries

A relational database is a collection of relations, each corresponding to a database predicate. Each relation R is a collection of tuples T_i satisfying the corresponding predicate *R*, i.e., $R(T_i)$ is TRUE. An intentional (derived) predicate is a predicate defined in terms of the database predicates. Let *V* be an intentional predicate that denotes a violation of the database Integrity Constraint *IC*, (i.e., *V* is the negation of *IC*). Efficient computation of *V* is critical in detecting semantic violations caused by erroneous database update operations. An integrity constraint is the primary tool of integrity maintenance system specifying a condition that should be satisfied by the database. A database update transaction is defined as a collection of insertions into and deletions from the database. Update Operation Repair Rules (*UORRs*) are defined as the rules that response to constraint violations. These rules have to take repair actions in the case of exploring errors. Update Operation Repair Rules have the following elements:

- The *event* is a set of pairs (u,r) with $u \in \{INS, DEL, UPD\}$ and *r* a relation name. The event specifies the update operations that may violate the constraint.
- The *condition* is a declarative specification of the integrity constraint. It describes the condition that should be met by the database in the formalism used for integrity constraints.
- The *repair action* is an extended relational algebra program, which specifies the actions to be taken if the *condition* of the rule is not satisfied by the database.

The following definitions which are related to the time spent in satisfying a given *UORRs* are used in the rest of this paper.

Definition 1: The total processing time to fire a rule (*FT*) is the total time of each processing unit participating in executing the rule.

Definition 2: (FC) is the total communications time between the demander and all the related subscribers participating in executing an update operation that fires the appropriate rules i.e. $FC = \sum_{x=0}^{n} tx$, where tx is the required time for the demander to send the request to subscriber *x*.

Definition 3: Total Rule Execution Time *(FTRET)* is the total of both processing time *(FT)* and communication time *(FC)* i.e. $FTRET = FT + FC$.

Definition 4: Quorum (*Q*) is the number of partitions accessed during the execution of a rule.

Lemma 1: Let a relation *P* with n attributes $a_1, a_2, \ldots, a_n$ is partitioned into *m* partitions according to attribute a_2. Assume that an *UORR* with condition c requires accessing some attributes from relation *P* then:

1. if the repair action is based on attribute a_2, this will reduce *FT*, *FC* and consequently *FTRET* compared to the repair action which is based on attributes other than a_2 .
2. if the repair action is based on attribute other than a_2, this will increase *FT*, *FC* and consequently *FTRET* compared to the repair action which is based on attribute a_2 .

Proof: In the first condition the requested tuple for repairing is directly located since the condition is based on a_2 and therefore *Q* is 1. So the process time *FT* will not have delay time since there will be only one subscriber replying to the demander. Therefore *FC* will be low and consequently *FTRET*. In the second condition the requested tuple for repairing is not directly located since the condition is not based on attribute a_2 and therefore $Q > 1$. The process time *FT* will have delay time since more than one subscriber will have to reply to the demander. Therefore *FC* will be high and consequently *FTRET*. □

Throughout this paper the same example Job Agency database is used, as given below. This example is taken from [10] and used in [3].

Person (pid, pname, placed, area_code, state)
Company (cid, cname, totsal)
Job (jid, jdescr)
Placement (pid, cid, jid, sal)

Application (pid, jid)
Offering (cid, jid, no_of_places, qid)

4. Physical Partitioning

Traditionally, the physical partitioning methods such as horizontal partitioning have been used to partition relations into separated subgroups of tuples, which have similar domains in a determined attribute group, each subgroup called partition. Once the tables are partitioned into fragments, it is important that the given algorithms are applied to the mechanisms that recover the information [5, 7]; that is to say, that the *UORRs* would know where to locate the target partition, once the condition implied in the rule is satisfied. In physical partitioning, locating the set of group of tuples should be requested according to the field which the table was partitioned, otherwise the locating algorithm will fail to reduce the time for locating the information required for executing the update operation and consequently firing the appropriate rules. The main partitioning objective is to reduce the time by applying an operation on specific tuples instead by applying of all tuples in a relation. One restriction of such kind of partitioning is that specifying the partitioning policy must be taken when the table is created. This includes the selection of attribute(s) as the criteria to partition the table. This means that once an attribute(s) has been selected and the table has been partitioned, no other partition policy can be applied to the table.

Example 1
Suppose *Person* relation is horizontal partitioned into *n* partitions according to *area_code* attribute. If we have the following update operation:

```
when UPD (personi (pad, pname, placed,
  area_code, state))
 if area_code=211  then
   state = "Selangor"
```

where person*i* (pid, pname, placed, area_code, state) is the partition created with the condition area_code= 211. Any attempt to update (UPD) the *person* table with area_code=211, will execute the above update operation. The quorum for this update operation is 1. Therefore, the lowest quorum is obtained for this example.

If the update operation is stated as follows:

```
when UPD (personi (pid, pname, placed,
  area_code, state))
 if state = "Selangor" then
   area_code  = 211
```

where i =1 to n and the same condition as the above example is used to partition the person relation. Any attempt to update (UPD) the *person* table with the condition state = "Selangor" will execute the above update operation to all of the partitions and therefore the quorum for this update operation is n.

Theorem: Let $R\ (a_1, a_2, \ldots, a_n)$ denotes a relation partitioned horizontally by the attribute *a1* into different partitions $r_1, r_2, \ldots, r_m$. Execution of an update operation *UO* firing a rule *Rule1* using attributes $(a_2, a_3, \ldots, a_n)$ will cause a high *FTRET* which negatively effect the execution of the update operation by leading to more extra response time and cost.

Proof: Executing an update operation *UO* on a relation $R\ (a_1, a_2, \ldots, a_n)$ firing the rule *Rule1* causes the rule parser to scan Q partitions $r_1, r_2, \ldots, r_m$ of the relation where Q >1. The requested tuple for checking whether the condition of the rule is satisfied or not is not directly located since the condition is not based on attribute a_1 and therefore Q >1. This will lead to more extra response time and cost as shown in Figure 1. □

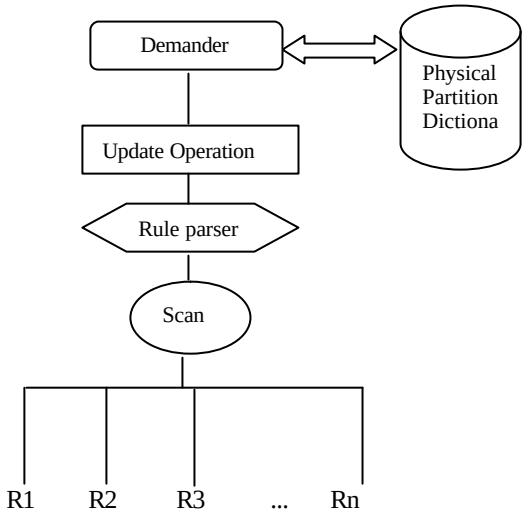

Figure 1. Physical partitioning method.

5. Virtual Rule Partitioning Method

Virtual Rule Partitioning (VRP) concentrates in generating, as solicited by the user at any time *t1*, a Virtual Partition (VP) of tuples *Ti* satisfying one or several conditions specified in *UORR*. *VP* will be emitted in different times *t2* $(\forall t2)(t2>t1)$. *VRP* is applicable to parallel database systems that work in architecture of type nothing share. *VRP* main objective is to reduce the *FTRET* used to repair erroneous database state.

The set of *UORR* specifications in our approach is defined as follows. *event* is a specific update operation. *condition* is an integrity constraint specification, and *repair action* is an extended relational algebra repair action. Then the following construct is a *UORR*s specification:

```
 WHEN event
CREATE VIRTUAL PARTITION
Vpname
ON predicate
```

attributes (X_1, X_2,…, X_n)
(RANK (X_i)
Between ('cv_1', 'cv_2' partition: 0),
Between ('cv_3', 'cv_4' partition: 1),
.
.
.
Between ('cv_{n-1}', 'cv_n' partition: n-1),
Default (partition: n))

if not condition
then repair action

where event={INS, DEL, UPD}
cv_1, cv_2, …, cv_n are constant values.
repair action={INS, DEL, UPD}

Example 2

CREATE VIRTUAL PARTITION
Vp_no_of_places ON offering
ATTRIBUTES (no_of_places)
(RANK (no_of_places)
Between ('1', '10' partition: 1),
Between ('11', '20' partition: 2),
Default (partition: 3))

CREATE VIRTUAL PARTITION
Vp_jid ON offering
ATTRIBUTES (jid)
(RANK (jid)
Between ('1', '5' partition: 1),
Between ('6', '10' partition: 2),
Default (partition: 3))

Based on the above example the system will create six virtual partitions. These partitions will contain an organization strategy according to the values of the *no_of_places* and *jid* attributes.

when UPD (offering (cid, jid, no_of_places, qid))
if not ("X)(X Î (offering) Þ X.no_of_places= 0)
then
delete (offering (cid, jid, no_of_places, qid),
$\sigma_{no\_of\_places<0}$(offering));

when UPD (offering (cid, jid, no_of_places, qid))
if not ("X)(X Î (offering) Þ X.jid=3)
then
UPD (X.jid=3);

Such that in both rules Q =1. The virtual Update Operation Repair Rule partitioning mechanism will result in *Q* =1, where *n* is the number of nodes, reducing *FT* and *FC* and consequently *FTRET*.

The flexibility of creating or eliminating the *VRP* at any time depends on the user request is one of the major benefits of the proposed methodology. The proposed method concentrates on locating the nodes to the rules where the desired partition requested by the rules for repairing the erroneous state. So the mechanism is not restricted to a specific physical partitioning.

Example 3
Following the intentional rule:

when INS (Application (pid, jid))
if not "(X)(X Î Application Þ $(Y)(Y Î Person and X.pid=Y. pid))
then INS (Person (pid, pname, placed, area_code, state))

which implies that when a *person* pid applies for a job (insert into *application* table) and the information of the person is not in the *person* relation then one of the repair action that can be performed is to insert the information of the person into the *person* relation.

Giving the constraint *C*:
"*Preventing person pid to apply for a job without being inserted in the person relation***"**

Consider an update operation:
T=INS (Application (Pid, Jid))

Supposing that person who has the number *Pid* has no tuple in *person* relation, the update operation will violate the constraint *C* and can be repaired by:

WHEN INS (Application (Pid, Jid))
CREATE VIRTUAL PARTITION
Vpname
ON Person
ATTRIBUTES (Pid)
(RANK (Pid)
Between ('1', '100' partition: 1),
Between ('101', '200' partition: 2),
Between ('201', '300' partition: 3),
Between ('301', '400' partition: 4),
Between ('401', '500' partition: 5),
Default (partition: 6))
if not "(X)(X Î application Þ $(Y)(Y Î Vpname and X.pid=Y.pid)
then INS (Person (pid, pname, placed, area_code, state))

The lowest quorum is the most important request for the demander update operation to be executed properly. Referring to the example in section 4, the first update operation insure that accessing the physical partitioning of the table will result in reducing the total rule execution time since the *person* table physically partitioned using the *area_code* attribute. When the table also has *VPs* according to some attributes, all of *VPs* will be evaluated to choose the best strategy by comparing the quorum of each of the *VP* to select the lowest one. On the other hand, if the rule fired through an update operation affecting the *person* table using attribute other than *state* to locate a part of data for updating, then accessing the physical partitioning will

be excluded and substituted by the virtual partitioning strategy.

Virtual rule partitioning strategy has advantages over physical partitioning and parallel processing is a proper environment to execute update operations with repair rules. There are some constraints, which restrict using such strategy. The idea of demander and subscribers is recommended to update operation with a very maximum workload. There is a communication between the demander and other subscribers to execute the global operation. This update operation with the fired rules will take more time when there is just little data affected by the update operation. This is because every subscriber will have to communicate with the demander informing it that it hasn't find tuples, which accommodate the condition. So centralization with no partitioning strategy either physical or virtual will be more advisable in such cases.

6. Results

We have run different experiments using simulation system with the following parameter settings:

N: Number of Processors.
MIT: Mean Interarrival Time.
MET: Mean Execution Time.
MRDT: Mean Repair Data Time.
NUO: Number of Update Operation.

where Mean Interarrival Time is the average time of sending update operations to the system. The Mean Execution Time is the average time of executing every update operation, while Mean Repair Data Time is the average time of repairing the erroneous state, which occurs when an update operation is executed.

Three experiments run 1000 update operations on 5 processors (see Tables 1, 2, 3 and Figures 2, 3, 4). As we can see the proposed method has reduced the update operation delay in the queue. This result is due to the fact that the demander could not determine where the tuples with the specified condition are located in the physical partitions. This state occurs when the tuple is partitioned according to an attribute other than the attribute in the specified condition.

Table 1. First experiment.

INPUTS		OUTPUTS		
			VRP	PhRP
MIT	1 second	ADQ	0.869	2.565
MET	2 seconds	ANQ	0.78	2.224
MRDT	4 seconds	SU	3.899	4.401
NUO	1000	TSE	1113.092	1163.126

VRP: Virtual Rule Partitioning, PhRP: Physical Rule Partitioning, ADQ: Average Delay in Queue, ANQ: Average Number in Queue, SU: Server Utilization, TSE: Time Simulation Ended.

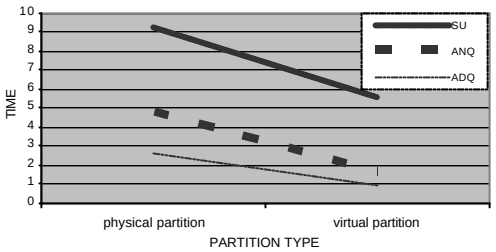

Figure 2. Repair times used for a specific *UORR* (1st experiment).

Table 2. Second experiment.

INPUTS		OUTPUTS		
			VRP	PhRP
MIT	1 second	ADQ	1.588	3.691
MET	2 seconds	ANQ	1.483	3.113
MRDT	6 seconds	SU	4.368	4.62
NUO	1000	TSE	1113.092	1163.126

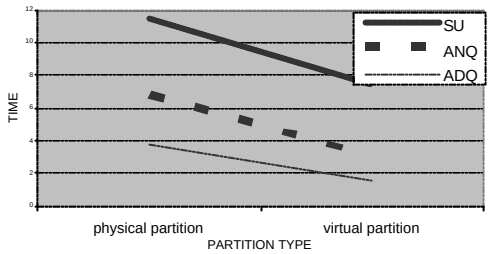

Figure 3. Repair times used for a specific *UORR* (2nd experiment).

Table 3. Third experiment.

INPUTS		OUTPUTS		
			VRP	PhRP
MIT	1 minute	ADQ	2.759	3.415
MET	4 minutes	ANQ	2.288	3.093
MRDT	2 minutes	SU	4.508	4.508
NUO	1000	TSE	1105.28	1207.812

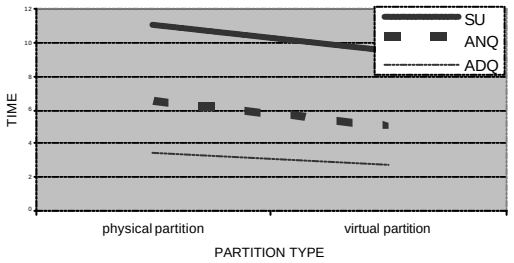

Figure 4. Repair times used for a specific *UORR* (3rd experiment).

7. Conclusion

We have shown that virtual rule partitioning as a new method of partitioning allows us to diminish the time of processing used by firing some *UORRs*. The physical partitioning methods help to a great extent to

speedup the execution of rules. These methods depend on the creation of the tables, which specify the partitioning policy that is taken when the table is created. This includes the selection of an attribute as the criteria to partition the table. Total execution time of the *UORRs* will highly affected by the form in which the tables were partitioned. Our strategy *VRP* is not concern by the way the table is partitioned. *VRP* can be used in parallel database systems, and we have to take into account that parallelism is not always the proper way to execute update operations. Parallel database systems are more advisable to be used in a maximum workload means that the *UORRs* must be fired to a huge amount of data to scan it. Applying the proposed approach with little amount of data will result in extra time between the demander and subscribers in different nodes just to tell the demander that there is no tuples satisfying the condition.

References

[1] Grefen P. W. "Integrity Control in Parallel Database Systems," *PhD Thesis*, University of Twente, Netherlands, October 1992.

[2] Hanandeh F. A., Ibrahim H., and Muda Z., "A Strategy for Semantic Integrity Maintenance for Parallel Relational Database Systems," *in Proceedings of the 2002 International Arab Conference on Information Technology (ACIT'2002)*, Qatar, vol. 2, pp. 714-719, December 2002.

[3] Ibrahim H., "Extending Transactions with Integrity Rules for Maintaining Database Integrity," *in Proceedings of the International Conference on Information and Knowledge Engineering (IKE'02)*, USA, pp. 341-347, June 2002.

[4] Ibrahim H., "Semantic Integrity Constraints Enforcement for Distributed Database," *PhD Thesis*, University of Wales Cardiff, June 1998.

[5] Jaime A. A., "Mixed Fragmentation Strategy in a General Purpose Parallel Databases System," *Second National Encounter of Computation ENC99*, Mexican Society of Sciences of the Computation and Mexican Society of Artificial Intelligence, Autonomous University of the State of Hidalgo, Mexico, September 1999.

[6] McCarroll N. F., "Semantic Integrity Enforcement in Parallel Database Machines," *PhD Thesis*, Department of Computer Science, University of Sheffield, Sheffield, UK, May 1995.

[7] Ozsu M. T. and Valduriez P., *Principles of Distributed Database Systems*, Prentice Hall, 1991.

[8] Seong-Jin P. and Bair D., "A Data Allocation Considering Data Availability in Distributed Database Systems," *International Conference on Parallel and Distributed Systems*, Korea, pp. 708-713, 1997.

[9] Thomas J. and Waston R., "A Fractional Data Allocation Method for Distributed Databases," *in Proceedings of the Third International Conference on Parallel and Distributed Information Systems*, NY, USA, pp.168-175, September 1994.

[10] Wang X. Y., "The Development of a Knowledge-Based Transaction Design Assistant," *PhD Thesis*, Department of Computing Mathematics, University of Wales College of Cardiff, Cardiff, UK, 1992.

Feras Hanandeh obtained his BSc in computer science from Yarmouk University, Jordan in 1993, MSc in computer science from University Putra Malaysia, Malaysia in 2000. Currently, he is a PhD candidate at University Putra Malaysia. His research interest includes parallel databases and logic programming. He has published a number of papers related to these areas.

Hamidah Ibrahim obtained her PhD from the University of Wales, Cardiff in 1998. Currently, she is a lecturer at the Faculty of Computer Science and Information Technology, University Putra Malaysia. Her research areas include distributed databases and knowledge based system. She has published a number of papers related to these areas.

Ali Mamat is an associate professor in the Computer Science Department at University Putra Malaysia. He obtained his PhD in 1992. His research interest includes databases, logic programming and knowledge base.

Rozita Johari obtained her BSc in computer science and mathematics from Pittsburg State University, Kansas in 1986, MSc in computer science from Illinois Institute of Technology, Chicago in 1987 and completed her PhD at University Putra Malaysia.