Distributed Rule Monitoring in Active Databases and Its Performance Analysis

Ing-Miin Hsu, Mukesh Singhal, Ming T. Liu
Department of Computer and Information Science
The Ohio State University
2036 Neil Avenue Mall
Columbus, Ohio 43210

August 10, 1992

Abstract

Monitoring rules in a distributed active database involves three design issues: how to decompose rules, how to distribute rules to sites, and how to evaluate distributed rules correctly. In this paper, we study these three issues for complicated rules, which are complex and time-consuming to evaluate. We propose a new relational operator, AND, and the associated algebraic manipulations of this operator to find independent parts of rule query, which can be distributed among sites. Due to problems introduced by geographical distribution of sites in a distributed system, such as communication delay and lack of a global clock, correct evaluation of distributed rules is not an easy task. We propose a distributed evaluation algorithm which guarantees the correctness of the evaluation result of the distributed rule by collecting consistent local results from sites to form the global view. The message count and the response time of this distributed evaluation algorithm are analyzed for triggering updates.

1 Introduction

Inference power is one of the most sought-after features in the next generation database management systems (DBMS's) [7,22]. To achieve non-procedural inference power, production-style rules have been introduced into relational DBMS's [10,27]. Rules are used not only as view definitions but also as triggers or alerters [5,27]. When rules are supported as triggers or alerters to implement traditional DBMS functions, such as integrity constraints and protection, database systems change from passive to active in the sense that a database system will take predefined action by itself whenever the database is transiting into certain predefined states.

In general, a rule has three parts, *a triggering event*, *a rule condition*, and *a rule action*. When a triggering event occurs, the rule condition needs to be evaluated. If the rule condition is true, the corresponding rule action should be executed. Otherwise, no action should be taken. Evaluating every possible rule condition upon every triggering event in the system places a tremendous performance requirement on the success of the active databases. Distributing rule monitoring over multiple sites offers an opportunity to parallelize this evaluation process. In addition, distributed DBMS's are emerging for reasons such as local autonomy, incremental growth, reliability, and flexibility. Along with these benefits come the problems introduced by distributed database systems, for example, global consistency, data fragmentation, and the lack of a global clock. Therefore, the problem of rule monitoring in a distributed active database needs to be investigated to successfully integrate rules into distributed DBMS's.

The design of distributed rule monitoring in active databases involves three issues:

1. how to decompose a rule condition into several parts to exploit parallelism;

2. how to distribute decomposed rule conditions and base data to sites so that rule evaluation can be carried out in parallel in multiple nodes; and

3. how to collect evaluation results of decomposed rule conditions from multiple nodes in a distributed environment so that the correctness of the distributed evaluation is guaranteed.

The first and second issues are highly interrelated and have been addressed previously [6,8,30]. The third issue, however, has not been addressed before, nor has the problem of distributed rule monitoring yet been addressed as a whole.

The remainder of the paper contains seven sections, beginning with a discussion of issues in distributed rule monitoring. Section 3 follows with examples of sample rules in a stock trading database system using PRS II syntax [28], which will be used throughout this paper. Section 4 introduces a rule decomposition algorithm which uses algebraic manipulations of relational operators to decompose a rule condition into independent relational expressions. In Section 5, a distribution algorithm to adapt rule distribution to base relation distribution is discussed. Section 6 proposes a distributed rule evaluation algorithm that guarantees the correct evaluation of a distributed rule condition and Section 7 analyzes the performance of this distributed evaluation algorithm. Finally, Section 8 concludes with final remarks.

2 Issues in Distributed Rule Monitoring

To utilize the parallel processing power offered by a distributed system, the initial issue in distributed rule monitoring is determining how to decompose a rule into several subrules, which can be evaluated on multiple sites in parallel. The approaches proposed in the past may be considered to decompose a rule at various levels, namely, at rule set level, at rule level, at relation level, at relational operator level, and at task level [4,11].

- At the rule set level, individual rule is not decomposed. Intead, parallelism is achieved by dividing the rule set into multiple partitions, each of which consists of multiple rules. This approach is adopted in most of the research on multiple rule firing for parallel production systems [11,13,15,17,20,21,24,26].

- At rule level, copy-and-constraint technique [25] creates multiple copies of the same rule, where each copy is constrained to match a subset of the data. In deductive databases, an equivalent idea using *discriminating predicates* is also independently reported in [8,30].

- At relation level, rules are decomposed into single-relation subrules in [12] for processing recursive queries in parallel.

- At relational operator level, each test node of the Rete algorithm can be considered as performing a relational operation, selection or join. In MAPPS [18], different test nodes of the Rete algorithm are distributed to different processing elements to achieve parallelism.

- At task level, the parallel Rete algorithm proposed in [9] and the DRete proposed in [14] parallelize the rule monitoring by decomposing each test node into tasks and distribute each task independently. Each task is identified by a test node with one set of data tuples. Work has also been done to parallelize individual relational operators, most notably the join operator [2].

After a rule is decomposed, it is important to address the question of how to match decomposed subrules to sets of base relations in order to distribute both of them to sites. Three possible distribution philosophies are (1) having a fully replicated database for the base relations [30], (2) adapting a base relation distribution to a rule distribution [1,6], and (3) adapting a rule distribution to a base relation distribution. With a fully replicated database, the distribution algorithm has the highest freedom in choosing a site for each subrule, and the problem with the first decomposition issue is simplified. Adapting a base relation distribution to a rule distribution is popular with approaches developed for parallel environment and has been investigated in [1,6,11,19,21]. However, no one has proposed a possible rule distribution strategy that does not require reorganization of the current base relation distribution. Since a base relation distribution is much less adaptable than rules in most distributed databases, this approach needs to be investigated further.

The third issue is determining how to collect results of subrule evaluation from multiple sites to form a global view of the entire rule condition. The criterion in this issue is correctness and consistency. A distributed evaluation algorithm is correct if and only if, first, it reports all rules whose conditions have occurred (i.e. whose conditions are true globally), and, second, it does not report rules whose conditions have not occurred. To guarantee the correctness of a global result, a consistent set of local results must be collected and combined. Traditional approaches in determine global state of a distributed system, such as global snapshots [3] and logical clocks[16], are not adequate in determining a consistent rule evaluation result. Global snapshots are suitable in detecting stable conditions, which stay unchanged once formed. However, rule conditions change dynamically since database states are changed constantly due to database updates. If a rule condition is not detected during the time window when the rule is true, it might not be detected at all. On the other hand, logical clocks force an arbitrary ordering among events that happen on different sites, which might cause false or no detection of rule condition satisfaction.

This paper addresses these three issues together for complicated nonrecursive rules, which not only involve multiple relations, but also are time-consuming to evaluate and, moreover, have never before been studied in a distributed environment. Following the examples in Section 3, proposals are presented for these three issues. First a decomposition scheme is proposed that lies between relation level and relational operator level. The goal of the decomposition is to find independent parts of a rule so that these parts can be distributed to different sites of the system according to the current distribution of the base relations. No special assumptions and/or arrangements about the base relation distribution need to be made here. A distribution algorithm is then proposed for this decomposition scheme. Finally, a distributed evaluation algorithm, which guarantees the consistency and correctness of the global evaluation result, is proposed and its performance is analyzed.

3 Examples

Example 1: Assume a database system at a stock exchange with the following relational schema; all the relation and attribute names are fairly self-explanatory.

STOCKS(name, class, district, open, current, high, low,...)
INDEX(name, open, current, high, low, ...)
BUYERS(tax_id, name, class, ...)
TRANSACTIONS(trans_no, name, shares, price, seller, buyer, ...)

Using the syntax proposed in the revised POSTGRES rule system [27,28], the following are some possible rules that can be defined for this database. Rule R1 updates the Dow Jones Industrial Average of 30 Industries (DJIA) whenever one of the 30 major stocks trades with a new price. Rule R2 alarms the stock exchange manager whenever the DJIA falls more than 100 points from today's opening index. Rule R3 is used to update today's high whenever stock price exceeds the old value. Rule R4 disallows program trading on major stocks whenever the DJIA fluctuates more than 50 points in a day.

```
on     replace to STOCKS.current                      (R1)
where  S.class = major
do     replace INDEX (current = sum(S.current))
using  S in STOCKS
where  INDEX.name = "DJIA"
and    S.class = major
```

```
on      replace to INDEX.current                                  (R2)
where   INDEX.name = "DJIA"
and     INDEX.open - INDEX.current > 100
do      alarm_mgr("DJIA")

on      replace to STOCKS.current                                 (R3)
where   STOCKS.current > STOCKS.high
do      replace STOCKS (high = new.current)

on      append to TRANSACTIONS                                    (R4)
using   I in INDEX
using   S in STOCKS
using   U in BUYERS
where   I.name = "DJIA"
and     (I.current - I.open > 50 or I.open - I.current > 50)
and     TRANSACTION.name = S.name
and     S.class = major
and     ((TRANSACTION.buyer = U.name
or      TRANSACTION.seller = U.name)
and     U.class = prog)
do      refuse append TRANSACTIONS                                 □
```

4 Decomposition of Rule Queries

4.1 Preliminaries

The goal of evaluating a rule condition is to determine whether the rule condition has occurred, which can be represented as a true/false value, and, when it occurs, to pass to the rule action a subset of the derived data, which is called the *binding* [5] between a rule condition and its corresponding rule action. Since rule conditions introduced in relational DBMS's have taken the form of database queries, a rule condition restricted to its binding with the rule action can be easily converted to a relational expression. Optimization of a relational expression by algebraic manipulations of its corresponding relational operator tree has been studied in detail [29]. The principle idea is to move selections and projections as far down the tree as possible. Our decomposition algorithm will follow the same principle. Moreover, part of a rule condition might simply be an existential qualification check, which does not use data generated from the triggering event or generate any data for the rule action. The existence of these independent parts of a rule condition offers an excellent opportunity to process a rule in a distributed environment with minimum communication overhead.

Let a **rule query** be the relational expression derived from a rule condition projected to extract data required by the binding with the rule action. The outcome of evaluating a regular database query is a relation that satisfies the relational expression. The outcome of evaluating a rule query, however, is a relation satisfying the relational expression and a logical value. A relational expression is considered logically true if the derived relation is not empty; otherwise, it is considered false. A new **AND operator** is introduced to capture this difference between rule queries and regular database queries. The AND operator is a binary operator with relational expressions as operands. The relational value of an AND expression is a relation that consists of two disjoint relations, each corresponding to an operand relation. The logical value of an AND expression is true when both of the operand expressions are true. The symbol Λ will be used to denote the AND operator in the rest of this paper, while conventional notations are used for existing relational operators, such as selection(σ), projection(π), Cartesian product($\times$), join($\bowtie$), union($\cup$), intersection($\cap$), and set difference($-$).

The following subsection presents the algebraic laws manipulating the AND operator, while laws regarding the algebraic manipulations of regular relational operators are included in Appendix A.

4.2 Laws involving AND operator

Let E_i, for any i, stand for some relational expression; F or F_i for some condition; A_i, B_i, and C_i for some attribute names.

1. Identity laws for AND operator.

   If F does not involve E_2,

   $$\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \land E_2 \tag{1}$$

   If $F = F_1 \wedge F_2$, F_1 involves E_1 and E_2 only, and F_2 involves E_2 and E_3 only,

   $$\sigma_F(E_1 \times E_2 \times E_3) \equiv \sigma_{F_1}(E_1 \times E_2) \land \sigma_{F_2}(E_2 \times E_3) \tag{2}$$

2. Commuting selection and projection with AND operator

   If F only involves E_1,

   $$\sigma_F(E_1 \land E_2) \equiv \sigma_F(E_1) \land E_2 \tag{3}$$

   If $F = F_1 \wedge F_2$ and F_1 involves only E_1 and F_2 involves only E_2,

   $$\sigma_F(E_1 \land E_2) \equiv \sigma_{F_1}(E_1) \land \sigma_{F_2}(E_2) \tag{4}$$

   If $A_1, \ldots, A_n$ are attributes of E_1 but not of E_2,

   $$\pi_{A_1, \ldots, A_n}(E_1 \land E_2) \equiv \pi_{A_1, \ldots, A_n}(E_1) \land E_2 \tag{5}$$

   If $A_1, \ldots, A_n$ is a list of attributes of which $B_1, \ldots, B_m$ are attributes of E_1, and the remaining attributes, $C_1, \ldots, C_k$, are from E_2,

   $$\pi_{A_1, \ldots, A_n}(E_1 \land E_2) \equiv \pi_{B_1, \ldots, B_m}(E_1) \land \pi_{C_1, \ldots, C_k}(E_2) \tag{6}$$

   If $A_1, \ldots, A_n$ are attributes of both E_1 and E_2,

   $$\pi_{A_1, \ldots, A_n}(E_1 \land E_2) \equiv \pi_{A_1, \ldots, A_n}(E_1) \cap \pi_{A_1, \ldots, A_n}(E_2) \tag{7}$$

3. Associative law for AND operator.

   $$(E_1 \land E_2) \land E_3 \equiv E_1 \land (E_2 \land E_3) \tag{8}$$

4.3 Decomposition Algorithm

The decomposition algorithm by algebraic manipulations of relational operators follows; equations 1 through 8 governing the manipulations of the new AND operator are listed in the preceding subsection, and equations 9 through 26 governing the manipulations of the existing relational operators are included in Appendix A.

Input: A relational expression represented as a relational operator tree.

Output: An optimized *decomposable* relational expression represented as a relational operator tree.

Algorithm:

1. Separate each selection $\sigma_{F_1 \wedge \ldots \wedge F_n}(E)$ into the cascade $\sigma_{F_1}(\ldots(\sigma_{F_n}(E)\ldots)$ using equations 15 and 16.

2. For each selection, move the selection as far down the tree as possible using equations 18-22, 24, and 26.

3. Use the identity equations for AND operator, equations 1 and 2, to find independent expressions and introduce AND node into the tree. Then move the AND node as far up the tree as possible using equations 3-7. Multiple AND nodes can be consolidated into one with multiple child nodes as suggested in equation 8.

4. For each projection, move the projection as far down the tree as possible, using equations 5-7, 18-19, 23, and 25.

5. Combine cascades of selections and projections into a single selection, a single projection, or a selection followed by a projection using equations 15-19.

6. Group each child node of an AND or intersection operator and all its descendants into a subrule.

Example 2: The rule query R4 from Example 1 can be represented as the following relational expression from its definition, where a relation name preceded by Δ identifies the triggering relation. This expression is used as input to the above decomposition algorithm:

$$\pi_T(\sigma_{\substack{I.name = \text{"DJIA"} \wedge \\ ((I.current - I.open > 50) \\ \vee (I.open - I.current > 50)) \wedge \\ T.name = S.name \wedge \\ S.class = major \wedge \\ ((T.buyer = U.name) \vee \\ (T.seller = U.name)) \wedge \\ U.class = prog}} (I \times S \times \Delta T \times U))$$

Step 1 transforms R4 by separating the selection into a cascade:

$$\pi_T \, \sigma_{I.name = \text{"DJIA"}} \, \sigma_{(I.current - I.open > 50) \vee (I.open - I.current > 50)} \, \sigma_{T.name = S.name} \, \sigma_{S.class = major} \, \sigma_{(T.buyer = U.name) \vee (T.seller = U.name)} \, \sigma_{U.class = prog} (I \times S \times \Delta T \times U)$$

Step 2 pushes the selections down the tree as close to the base relation it refers to as possible, which transforms R4 into

$$\pi_T((\sigma_{I.name = \text{"DJIA"}} \, \sigma_{(I.current - I.open > 50) \vee (I.open - I.current > 50)} I) \times (\sigma_{T.name = S.name} \, \sigma_{(T.buyer = U.name) \vee (T.seller = U.name)} ((\sigma_{S.class = major} S) \times (\sigma_{U.class = prog} U) \times \Delta T)))$$

At step 3, R4 is transformed into the following expressions in sequence, using equations 1 and 2 to identify possible replacement of Cartesian product with AND operators and equation 5 to push AND operators upwards:

$$\pi_T(\sigma_{\substack{I.name\\=\\\text{“DJIA”}}}\ \sigma_{\substack{(I.current\\-\ I.open\\>50)\\\vee\\(I.open\ -\\I.current\\>50)}}\ I \,\Lambda\, \sigma_{\substack{T.name\\=\\S.name}}\ \sigma_{\substack{(T.buyer\\=\\U.name)\\\vee\\(T.seller\\=\\U.name)}}\ ((\sigma_{\substack{S.class\\=\\major}}\ S) \times (\sigma_{\substack{U.class\\=\\prog}}\ U) \times \Delta T))$$

$$\sigma_{\substack{I.name\\=\\\text{“DJIA”}}}\ \sigma_{\substack{(I.current\\-\ I.open\\>50)\\\vee\\(I.open\ -\\I.current\\>50)}}\ I \,\Lambda\, \pi_T(\sigma_{\substack{T.name\\=\\S.name}}\ ((\sigma_{\substack{S.class\\=\\major}}\ S)\times\Delta T) \,\Lambda\, \sigma_{\substack{(T.buyer\\=\\U.name)\\\vee\\(T.seller\\=\\U.name)}}\ ((\sigma_{\substack{U.class\\=\\prog}}\ U)\times\Delta T))$$

At step 4, equation 7 is used to push the projection operator down, which transforms R4 into the expression

$$\sigma_{\substack{I.name\\=\\\text{“DJIA”}}}\ \sigma_{\substack{(I.current\\-\ I.open\\>50)\\\vee\\(I.open\ -\\I.current\\>50)}}\ I \,\Lambda\, (\pi_T(\sigma_{\substack{T.name\\=\\S.name}}\ ((\sigma_{\substack{S.class\\=\\major}}\ S)\times\Delta T))\cap\pi_T(\sigma_{\substack{(T.buyer\\=\\U.name)\\\vee\\(T.seller\\=\\U.name)}}\ ((\sigma_{\substack{U.class\\=\\prog}}\ U)\times\Delta T)))$$

Figure 1 is the relational operator tree equivalent to the last expression, which can also be represented as the following subrules:

$$\sigma_{\substack{(I.name\,=\,\text{“DJIA”})\,\wedge\\((I.current\,-\,I.open\,>\,50)\ \vee\ (I.open\,-\,I.current\,>\,50))}}\ I \tag{$R4_1$}$$

$$\pi_T(\sigma_{T.name=S.name}((\sigma_{S.class=major}S) \times \Delta T)) \tag{$R4_{2_1}$}$$

$$\pi_T(\sigma_{(T.buyer=U.name)\,\vee\,(T.seller=U.name)}((\sigma_{U.class=prog}U) \times \Delta T)) \tag{$R4_{2_2}$}$$

$$R4_{2_1} \cap R4_{2_2} \tag{$R4_2$}$$

$$R4_1 \,\Lambda\, R4_2 \tag{$R4$}$$

□

Example 3: Likewise, rules R1, R2, and R3 from Example 1 can be transformed by the decomposition algorithm. R1 is transformed into one subrule

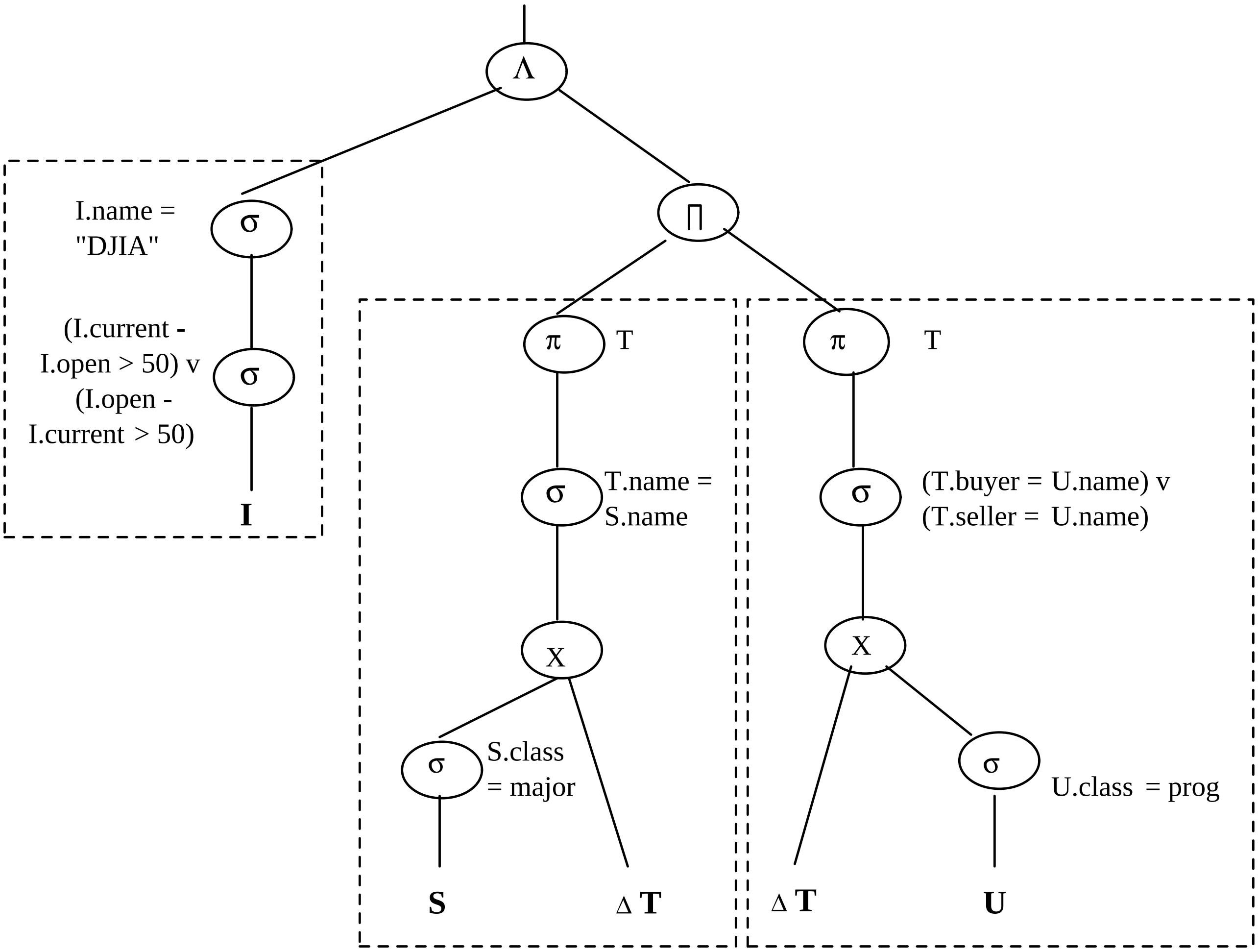

Figure 1: Decomposed Relational Operator Tree for Rule R4

$$\pi_{S.current}(\sigma_{S.class=major}\Delta S) \qquad (R1_1)$$

R2 is transformed into one subrule

$$\sigma_{(I.open-I.current>100)\ \wedge\ (I.name=\text{“DJIA”})}\Delta I \qquad (R2_1)$$

R3 is transformed into

$$\pi_{S.high,S.current}(\sigma_{S.current>S.high}\Delta S) \qquad (R3_1)$$

□

5 Distribution Algorithm

Assume a base relation is partitioned into some data segments horizontally, vertically, or a combination of both, and each data segment is distributed to a different site. A horizontal partition is identified by a *guard* selection, which must be satisfied by every tuple in the partition; while a vertical partition is identified by a *guard* projection, which contains a subset of the attributes of the original relation, and is shared by all tuples in the partition. The *distributed representation* of the original base relation is a union of the horizontal partitions, a natural join of the vertical partitions, or a combination of both.

Each subrule as derived from the previous decomposition algorithm can be further decomposed for distribution as follows. Replace each reference of the base relation in the subrule with its distributed representation. This replaced subrule can then be transformed by the same decomposition algorithm presented in the previous section to push selections and projections present in the original subrule down through the newly introduced union and join operators and their associated guard selections and projections. When combining an existing selection with the guard selection of some data segment, an entire subtree might be deleted due to contradiction of the two selection conditions. When combining an existing projection with the guard projection of some data segment, an entire subtree might also be deleted due to the empty intersection between the two sets of attributes. After this transformation, each subrule can be viewed as a tree with the union or join operator at the root with some subtrees as children. Each of these subtrees can then be distributed to site where the corresponding data segment, identified by the guard selection or projection, is allocated.

Example 4: Assume the STOCKS relation from Example 1 is distributed to 10 sites according to possible values of the *district* attribute. Then the distribution algorithm will distribute subrule $R3_1$ to these 10 sites, each as a subrule of

$$\pi_{S.high,S.current}(\sigma_{(S.current>S.high)\ \wedge\ (district=...)}\Delta S) \qquad (R3_{1_i})$$

Likewise, subrule $R4_{2_1}$ will be transformed into

$$\pi_T(\sigma_{T.name=S.name}((\cup R4_{2_{1_i}}) \times \Delta T)) \qquad (R4_{2_1})$$

$$\sigma_{(S.class=major)\ \wedge\ (district=...)}S \qquad (R4_{2_{1_i}})$$

□

6 Distributed Evaluation Algorithm

6.1 Basic Ideas

A distributed rule query is modeled as a relational expression tree, where each leaf node is allocated on a different site and an internal node might allocate on the same node as one of its children. In general, all nodes of the tree will calculate some relational expressions and pass the results to their parents; specifically, a leaf node derives its relation from some local base relation, while an internal node derives its relation from relations received from its children. The relation derived by the root node represents the binding between the rule condition and the rule action. When this relation is not empty, the root node is responsible for alerting the system and forwarding this data to the rule action. The evaluation algorithm in a distributed environment determines when the rule query is true in a timely and consistent way with minimum communication.

To evaluate subrules scattered among nodes in a distributed environment where no global clock is available and many tasks may happen simultaneously, coordination among nodes is necessary to guarantee the correct evaluation result. The *simultaneous* *region* approach [23] offers such a coordination scheme for a tree of nodes. Execution at each node is divided into regions, each of which is identified by a region number. Regions with the same region number on a group of sibling nodes are in effect in execution simultaneously and, thus, form a simultaneous region.

When a local update occurs at a node, the node will initiate a simultaneous region evaluation to propagate the change upwards. A node initiates a simultaneous region evaluation by sending its siblings *synchronization*(*sync*) messages, which contain nothing but its own region number, and advancing its own region by incrementing its region number by 1. Upon receiving a *sync* message, a node will advance its region if it has not done so already due to its own local update. Whenever a node increments its region number, it sends to its parent the latest evaluation result of its relational expression with the old region number in a *data* message, taking into account the local update that happens during that region.

An internal node will derive its relation with child relations from a simultaneous region, namely, with relations received from its children in *data* messages with the same region number. The new evaluation result is considered a local update to this internal node and triggers a simultaneous region evaluation among its siblings. Hence, an internal node maintains two separate region numbers, one for its child nodes and another one for its sibling nodes.

6.2 Procedure Pseudocodes

Assume a reliable underlying network which delivers the messages in the order sent and loses no message. Assume at each site *current_region_no* is initialized to 0 and *state_info* is initialized to reflect the proper subrule state.

Procedure local_update

/* Execute this procedure when a local update occurs */

```
Begin
    send sync message to all sibling nodes with current_region_no;
    state_info := eval(local expression);
    send data message to parent node with state_info, site_id, and current_region_no;
    current_region_no := current_region_no + 1;
End
```

Procedure recv_sync(sibling_region_no)

```
/* Execute this procedure when a sync message from a sibling node is received */
Begin
    while current_region_no ≤ sibling_region_no then
        Begin
        send data message to parent node with state_info, site_id, and current_region_no;
        current_region_no := current_region_no + 1;
        End
End
```

Procedure recv_data(child_state_info, child_id, child_region_no)

```
/* Execute this procedure when a data message from a child node is received */
Begin
    store child_state_info in a two-dimensional table indexed by child_id and child_region_no;
    if child_state_info of all child nodes are available for the child_region_no then
        Begin
        if parent = Null then
            Begin
            state_info := eval(child_state_info of all child nodes for the child_region_no);
            if state_info.status = True then
                Begin
                recognize the occurrence of rule condition;
                pass state_info to rule action;
                End
            else nop;
            End
        else call procedure local_update;
        release table space for all child_state_info with region number ≤ child_region_no;
        End
End
```

6.3 Argument for Correctness

Under the assumption of a reliable communication network, which loses no message and delivers messages in the order sent, the simultaneous region approach has the following properties [23]. (1) No region number will be skipped at each node because each node will be initialized with the same initial region number and will advance the region number successively. (2) For every region in a node, there is a corresponding, equally numbered, region in its sibling nodes.

From these properties, we can claim that (1) every rule query occurrence will be recognized and (2) no rule query which did not occur will be recognized. Claim 1 is achieved since every local update will be captured in a region. From property 2, there will be corresponding regions in its siblings. Further, each node will report its local result to the parent node when it advances its region. From property 1, a parent node will have the results from all of its children for a simultaneous region to derive its own result, and propagate it up the tree. Thus, if a rule query becomes true in some simultaneous region, this true status will be recognized by the root node. Claim 2 is achieved since a parent node will derive the global result with results from a simultaneous region only. So rule query occurrence will be recognized only when subrules are true in a simultaneous region.

7 Performance Analysis of the Distributed Evaluation Algorithm

7.1 Performance Model

A distributed rule query is modeled as a tree of nodes, where database updates, which change the state of the rule condition, may occur. The very last database update that makes the entire rule condition change from false to true or from true to false is termed

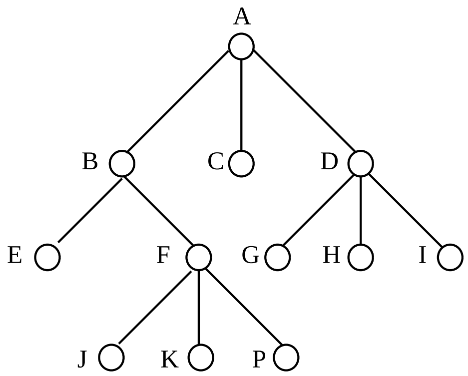

Figure 2: Sample Model of Distributed Rule

triggering update. The node where the triggering update occurs is termed *triggering leaf node*. After the triggering leaf node recognizes the occurrence of a triggering update, it initiates a simultaneous region evaluation among itself and its sibling in order to pass the information upwards. Its parent node, after a consistent set of evaluation results is collected from all its children, will recognize the occurrence of the triggering update, and thus will become a *triggering node* at its level and has the responsibility to trigger a simultaneous region evaluation among itself and its sibling to make the triggering update known to its parent. Eventually, the root node will recognize the occurrence of the triggering update and the changing state of the rule condition.

Definitions: The *level* of a node is the number of edges in the path from the node to the root node. Hence, the root node has a level 0. The *direct ancestors* of a node are all the nodes along the path from that node to the root. The *ancestors* of a node are all the nodes whose parents are direct ancestors of the node and whose levels are less than that of the node. The *siblings* of a node are all the nodes who have the same parent as the node.

Example 5: Figure 2 presents a sample tree of 12 nodes. Node J is at level 3. Nodes F, B and A are direct ancestors of J. Nodes E, F, B, C, D, and A are ancestors of J. Nodes K and P are J's siblings. □

For a triggering update to propagate to the root node, the siblings and the ancestors of the triggering leaf node are all involved in the evaluation process. Assume the triggering leaf node is at level ℓ with $m-1$ sibling nodes and n ancestor nodes. The message count and the response time for the triggering update to propagate to the root node are analyzed next.

7.2 Message Count

The number of messages exchanged for the triggering update to propagate to the root node from the triggering leaf node can be analyzed in terms of two types of messages, synchronization messages and data messages. Synchronization messages are fixed-sized, short, and are exchanged between sibling nodes. Data messages have varying sizes, have potential to be long, and are passed from children to parent node.

For a triggering update to propagate one level up from a node with $m-1$ siblings, $m-1$ synchronization messages and m data messages will be sent out for a simultaneous region evaluation at that level. The number of data messages sent is one plus the number of synchronization messages sent, because the triggering node does not send a synchronization message to itself.

For a triggering update originating at level ℓ to propagate up to the root, the above simultaneous region evaluation will be repeated ℓ times. Every sibling and every ancestor of the triggering leaf node will participate in this evaluation process. Every participated node except the root will send out a data message. Thus, the total number of data messages sent for the triggering update to propagate to the root is equal to the total number of ancestor nodes of the triggering leaf node minus 1 for the root node plus the total number of sibling nodes of the triggering leaf node plus 1 for itself. This number is $n-1+m$.

At each level, the number of synchronization messages required is one less the number of data messages required. Hence, the total number of synchronization messages required for the triggering update to reach the root node is equal to the total number of data messages minus the level of the triggering leaf node. This number is $n - 1 + m - \ell$.

7.3 Response Time

Response time will be measured as the time elapsed from the moment the triggering leaf node recognizes the occurrence of the triggering update to the moment the root node recognizes the changed status of the entire rule condition.

Let random variable Z denote the time elapsed from the moment the triggering node recognizes the occurrence of the triggering update to the moment it finishes the calculation incurred by the update. Assume this computation delay has an exponential distribution with means $\frac{1}{\lambda'}$. Let random variable Y denote the time elapsed from the moment the triggering node sends out the data message to the moment the parent node receives the data message. Assume this communication delay has an exponential distribution with means $\frac{1}{\lambda}$. From definition, the probability density functions (pdf's) for these two random variables are:

$$f_Y(y) = \begin{cases} \lambda e^{-\lambda y} & y \geq 0 \\ 0 & y < 0 \end{cases}$$

$$f_Z(z) = \begin{cases} \lambda' e^{-\lambda' z} & z \geq 0 \\ 0 & z < 0 \end{cases}$$

Let random variable V denote the time elapsed from the moment the triggering node recognizes the occurrence of the triggering update to the moment the parent node receives the data message from the triggering node. Then $V = Y + Z$. Let $f_{YZ}(y,z)$ denote the joint pdf for Y and Z. Assuming Y and Z are independently distributed, $f_{YZ}(y,z) = f_Y(y)f_Z(z)$. The pdf for V, $g(v)$, can be derived from $f_{YZ}(y,z)$.

(The detailed derivations of all equations in this section are left in Appendix B for the clarity of the presentation.) Note that $V \leq v$ whenever $Y \leq y$ and $Z \leq v - y$. Thus,

$$\begin{aligned} g(v) &= \int_0^v f_{YZ}(y, v-y)dy \\ &= \frac{\lambda\lambda'}{\lambda - \lambda'}(e^{-\lambda' v} - e^{-\lambda v}) \end{aligned}$$

From the pdf $g(v)$, the cumulative distribution function (cdf) for V, $G(v)$, can be derived.

$$\begin{aligned} G(v) &= \int_0^v g(t)dt \\ &= \frac{\lambda'}{\lambda - \lambda'}(e^{-\lambda v} - 1) - \frac{\lambda}{\lambda - \lambda'}(e^{-\lambda' v} - 1) \end{aligned}$$

The pdf and cdf of the time delay for the parent node to receive data message from a non-triggering node can be similarly derived. Let random variable X denote the communication delay for the synchronization message. Let random variable Z denote the computation delay at the non-triggering node. Let random variable Y denote the communication delay for the data message. Let U be the random variable that denotes the time elapsed from the moment the triggering update is recognized by the triggering node to the moment the parent of the triggering node receives the data message corresponding to the triggering update from a sibling of the triggering node. Then $U = X + Y + Z$. Assume X and Y have exponential distribution with the same parameter λ, Z has an exponential distribution with parameter λ', and they are mutually independent. Let f denote the pdf and F the cdf for U.

$$\begin{aligned} f(u) &= \int_0^u \int_0^{u-y} f_X(x) f_Y(y) f_Z(u-x-y)dxdy \\ &= \frac{\lambda^2\lambda'}{(\lambda - \lambda')^2}[e^{-\lambda' u} - (u(\lambda - \lambda') + 1)e^{-\lambda u}] \end{aligned}$$

$$
\begin{aligned}
F(u) &= \int_0^u f(t)dt \\
&= \frac{\lambda\lambda'}{(\lambda-\lambda')^2}e^{-\lambda u} - \frac{\lambda^2}{(\lambda-\lambda')^2}e^{-\lambda' u} + \frac{\lambda^2}{(\lambda-\lambda')^2} - \frac{\lambda\lambda'}{(\lambda-\lambda')^2} \\
&\quad + \frac{\lambda\lambda'}{\lambda-\lambda'}ue^{-\lambda u} + \frac{\lambda'}{\lambda-\lambda'}e^{-\lambda u} - \frac{\lambda'}{\lambda-\lambda'}
\end{aligned}
$$

For a node with m children, the time elapsed to pass the triggering update one level up to it is the time elapsed from the moment a triggering update is recognized by one of its children nodes to the moment this node receives the last data message for the simultaneous region corresponding to this triggering update from its children. Let random variable T denote this time delay. Let random variables $S_1, S_2, \ldots, S_m$ denote the time elapsed from the moment the triggering update is recognized by one of its children nodes to the moment the node receives the data message for the corresponding simultaneous region from child node i. Without loss of generality, assume child node 1 is the triggering child node. S_1 will have a cdf of G; while $S_2, \ldots, S_m$ will have cdf of F. Assume S_i's are mutually independent. The probability for the parent node to receive the last data messages incurred by the triggering update before t units of time is for it to receive a data message from every one of its children nodes before t units of time, i.e.,

$$
\begin{aligned}
F_T(t) &= \prod_{i=1}^{m} F_{S_i}(t) \\
&= G(t)F(t)^{m-1}
\end{aligned}
$$

Then the pdf for T can be derived.

$$
\begin{aligned}
f_T(t) &= \frac{\partial F_T(t)}{\partial t} \\
&= (m-1)f(t)F(t)^{m-2}G(t) + g(t)F(t)^{m-1}
\end{aligned}
$$

For any given λ, λ', and m, the mean response time to propagate a triggering update one level up is the expected value of $f_T(t)$, $E(T)$.

$$E(T) = \int_0^\infty t f_T(t) dt \tag{E1}$$

For a triggering node at level ℓ, let $E_i(T)$ denote the mean response time to propagate the triggering update one level up from level i to level $i-1$. Since the E_i's are mutually independent, the mean response time to propagate the triggering update from the triggering leaf node up to the root node is then $\sum_{i=1}^{\ell} E_i(T)$.

7.4 Performance Results

Since the integration in equation $E1$ does not have a closed form, the expected value of $f_T(t)$ is computed numerically using the following approximation:

$$E(T) = \sum_{n=0}^{\frac{Tlength}{\Delta T}} n(\Delta T)^2 f_T(n\Delta T)$$

ΔT is the width of the increment along the T axis and should be narrow enough in order to minimize the error around the peak of $f_T(t)$. $Tlength$, chosen to terminate the summation within an acceptable error range, is the maximal value that will be used to calculate $f_T(t)$ since $f_T(t)$ approaches zero asymptotically after the peak. In our numerical computation, we vary ΔT and $Tlength$ from 0.01 to 0.001 and from 0.9 to 4.2, respectively, to achieve an error margin of 1%.

Approximations of $E(T)$ are computed with parameters λ, λ', and m. Parameter λ is the inverse of the mean communication delay for one message, which includes transmission delay, propagation delay, and possible collision delay. Parameter λ' is the inverse of the mean computation delay in calculating a relational expression including disk access time. Parameter m is the number of sibling nodes plus 1 for the triggering node.

The mean response time to propagate a triggering update one level up from a group of m nodes is plotted with varying parameter values. Figure 3 plots the mean response

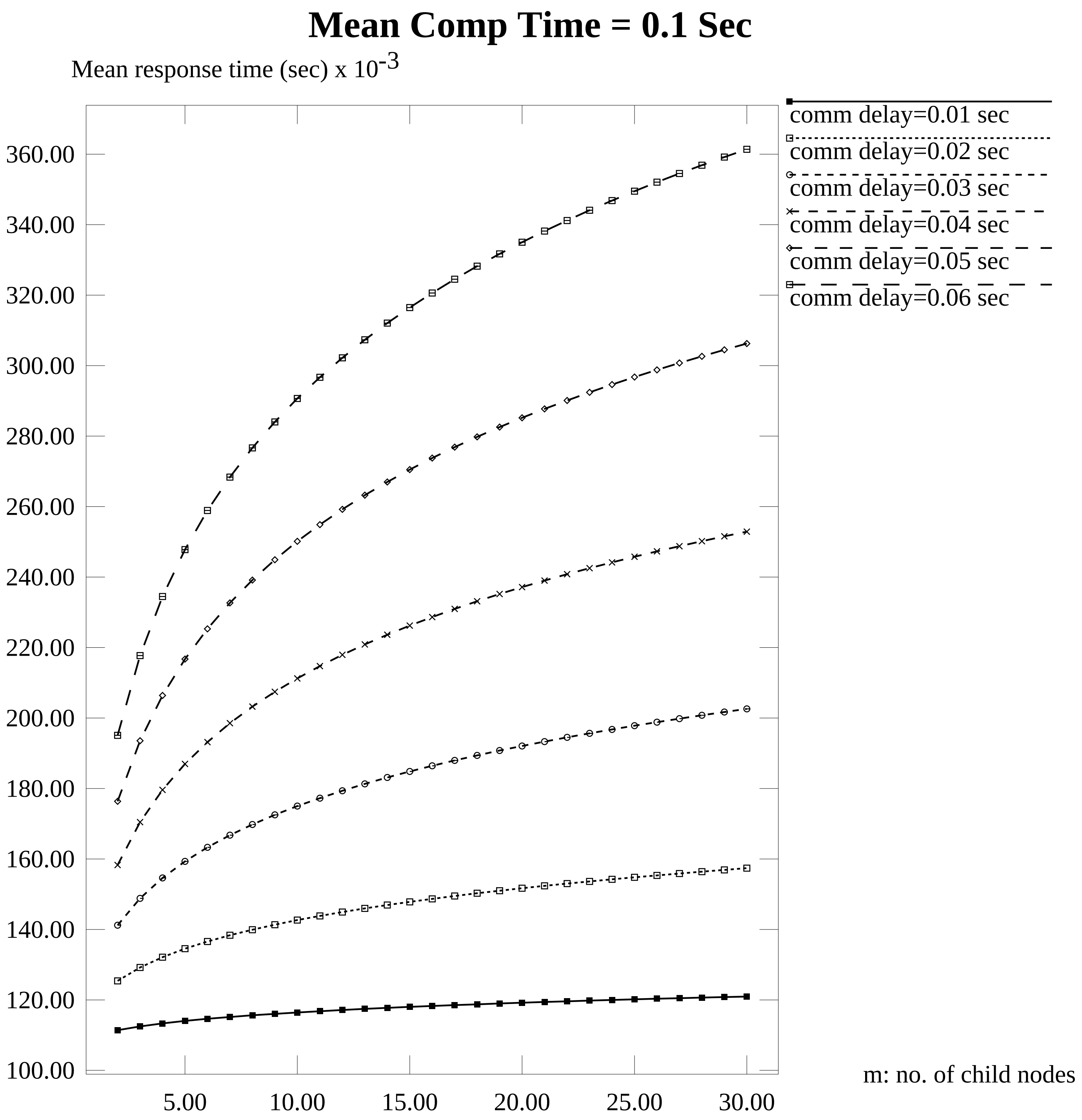

Figure 3: Mean response time for various mean communication delay values

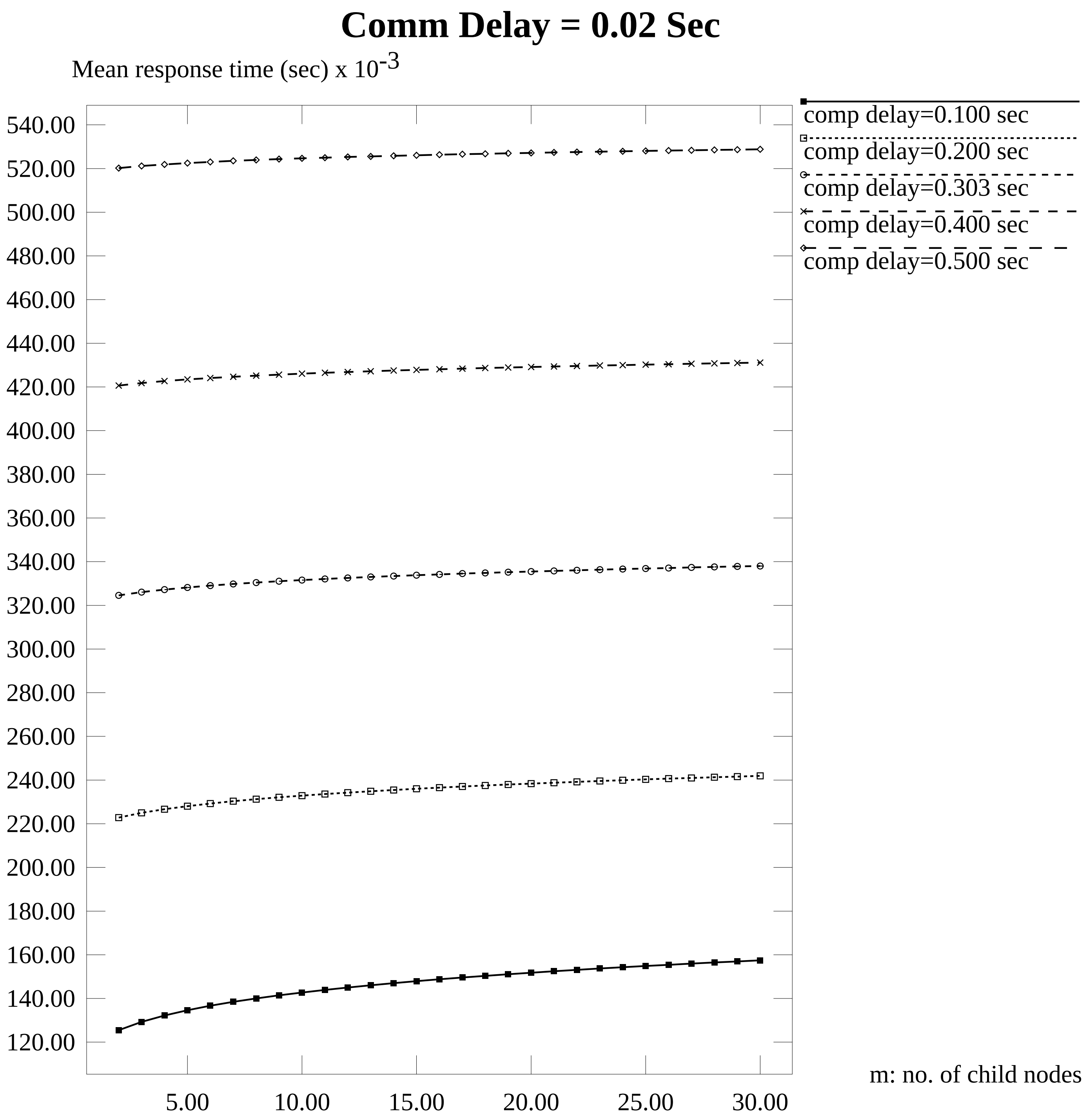

Figure 4: Mean response time for various mean computation delay values

time with mean computation delay fixed at 100 $m$sec and mean communication delay ranging from 10 $m$sec to 60 $m$sec. Figure 4 plots the mean response time with mean communication delay fixed at 20 $m$sec while mean computation delay ranging from 100 $m$sec to 500 $m$sec.

Figures 3 and 4 show that the mean response time is a monotonically increasing function with decreasing slopes relative to m. Let $E_m(T)$ denote the mean response time to propagate an update up a level of m nodes. The following inequalities can be observed for any $m > 2$:

$$\begin{aligned} E_m(T) &< E_{m-1}(T) + E_2(T) \\ E_m(T) &< 2 \times E_{\frac{m}{2}}(T) \\ E_m(T) &< E_{\frac{m}{2}}(T) + E_2(T) \end{aligned}$$

The right hand sides of the inequalities represent the mean response times to propagate an update two levels up from trees which correspond to various re-arrangements of the original m nodes. These inequalities indicates that the mean response time to propagate an update one level up is always smaller than the mean response time to propagate the update two levels up for any arrangement of the same number of nodes.

For a group of m nodes, there are many ways to arrange them into a tree. On one extreme is to place all of them at one level with a common parent. On the other extreme is to group every two of them together, introduce extra nodes as internal nodes, and build a binary tree of $\log m$ levels. For the second configuration of nodes, the number of internal nodes (n) increases linearly and the depth of the tree (l) increases logarithmically with m. With these increasing rates of n and l, the message counts of both *synchronization* and *data* messages will increase. The mean response time will increase for this configuration also from the previous observations of Figures 3 and 4.

Hence the analysis suggests that a fatter tree is better than a taller tree in terms of message count and response time for this distributed evaluation algorithm.

Nevertheless, the above analysis applies only to triggering updates, which toggle the global states of the rule condition from true to false or from false to true. For non-triggering updates, which do not change the global states of the rule condition, the response time is not an issue. The total message count, however, is different from the above analysis when non-triggering updates are taken into consideration. The message count for non-triggering updates can be minimized by limiting the number of nodes that need to participate in the distributed evaluation process, which probably suggests a taller tree is better than a fatter tree. However, the performance of non-triggering updates and its impact on the overall performance is still unclear at this point and will be the topic of future research.

8 Conclusions

In this paper, a decomposition scheme, a distribution algorithm, and a distributed evaluation algorithm have been proposed to process complicated rules in distributed active databases. The decomposition scheme uses a new relational operator, AND, to identify independent parts of a rule query and to facilitate the distribution of sub-rules according to the current base relation distribution. The distributed evaluation algorithm aims to derive a correct and consistent global rule evaluation result, which is difficult to achieve and has not been addressed previously due to the distributed environment. In this paper, the concept of simultaneous regions is applied to the decomposed rule query tree to achieve consistency and correctness in distributed rule evaluation. The message count and the response time of the distributed rule evaluation algorithm in processing triggering updates are analyzed. The probability density function of the response time is derived and the mean is approximated numerically.

The analysis of the distributed evaluation algorithm suggests a fatter tree is preferable to a taller tree in terms of response time and message count for triggering updates. The performance for non-triggering updates and its impact on overall performance is the topic of future research.

Appendix A

Algebraic laws in manipulating regular relational operators as introduced in [29] are presented in this appendix to complete the presentation of the decomposition algorithm. Conventional notations are used for these relational operators, such as selection(σ), projection(π), Cartesian product($\times$), join($\bowtie$), union($\cup$), intersection($\cap$), and set difference($-$). Let E or E_i, for any i, stand for some relational expression; F or F_i for some condition; A_i, B_i, and C_i, for some attribute names.

Laws involving joins and Cartesian Products

1. Commutative laws for joins and products.

$$E_1 \bowtie_F E_2 \equiv E_2 \bowtie_F E_1 \tag{9}$$

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1 \tag{10}$$

$$E_1 \times E_2 \equiv E_2 \times E_1 \tag{11}$$

2. Associative laws for joins and products.

$$(E_1 \bowtie_{F_1} E_2) \bowtie_{F_2} E_3 \equiv E_1 \bowtie_{F_1} (E_2 \bowtie_{F_2} E_3) \tag{12}$$

$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3) \tag{13}$$

$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3) \tag{14}$$

Laws involving selections and projections

3. Cascade of selections and projections.

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E) \tag{15}$$

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_2}(\sigma_{F_1}(E)) \tag{16}$$

If $A_1, \ldots, A_n$ is a subset of attribute names among $B_1, \ldots, B_m$,

$$\pi_{A_1, \ldots, A_n}(\pi_{B_1, \ldots, B_m}(E)) \equiv \pi_{A_1, \ldots, A_n}(E) \tag{17}$$

4. Commuting selections and projections.

If condition F involves attributes only from $A_1, \ldots, A_n$,

$$\pi_{A_1, \ldots, A_n}(\sigma_F(E)) \equiv \sigma_F(\pi_{A_1, \ldots, A_n}(E)) \tag{18}$$

If condition F involves attributes $B_1, \ldots, B_m$ that are not among $A_1, \ldots, A_n$,

$$\pi_{A_1, \ldots, A_n}(\sigma_F(E)) \equiv \pi_{A_1, \ldots, A_n}(\sigma_F(\pi_{A_1, \ldots, A_n, B_1, \ldots, B_m}(E))) \tag{19}$$

5. Commuting selections and projections with a Cartesian product.

If all the attributes mentioned in F are attributes of E_1,

$$\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2 \tag{20}$$

If $F = F_1 \wedge F_2$, and F_1 only involves attributes of E_1, and F_2 involves only attributes of E_2,

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2) \tag{21}$$

If $F = F_1 \wedge F_2$, and F_1 only involves attributes of E_1, but F_2 involves attributes of both E_1 and E_2,

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_2}(\sigma_{F_1}(E_1) \times E_2) \tag{22}$$

If $A_1, \ldots, A_n$ is a list of attributes of which $B_1, \ldots, B_m$ are attributes of E_1, and the remaining attributes, $C_1, \ldots, C_k$, are from E_2,

$$\pi_{A_1, \ldots, A_n}(E_1 \times E_2) \equiv \pi_{B_1, \ldots, B_m}(E_1) \times \pi_{C_1, \ldots, C_k}(E_2) \tag{23}$$

6. Commuting selections and projections with a union.

   If we assume that the attributes of E_1 and E_2 have the same names as those of E, or at least there is a given correspondence that associates each attribute of E with a unique attribute of E_1 and a unique attribute of E_2. Then we may write

$$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2) \tag{24}$$

$$\pi_{A_1,\ldots,A_n}(E_1 \cup E_2) \equiv \pi_{A_1,\ldots,A_n}(E_1) \cup \pi_{A_1,\ldots,A_n}(E_2) \tag{25}$$

7. Commuting selections with a set difference.

   With the same assumption as above,

$$\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2) \tag{26}$$

Appendix B

$$f_Y(y) = \begin{cases} \lambda e^{-\lambda y} & y \geq 0 \\ 0 & y < 0 \end{cases}$$

$$f_Z(z) = \begin{cases} \lambda' e^{-\lambda' z} & z \geq 0 \\ 0 & z < 0 \end{cases}$$

$$\begin{aligned}
g(v) &= \int_0^v f_{YZ}(y, v-y)dy \\
&= \int_0^v f_Y(y) f_Z(v-y)dy \\
&= \int_0^v \lambda e^{-\lambda y} \lambda' e^{-\lambda'(v-y)} dy \\
&= \lambda\lambda' e^{-\lambda' v} \int_0^v e^{-(\lambda-\lambda')y} dy \\
&= -\frac{\lambda\lambda'}{\lambda-\lambda'} e^{-\lambda' v} e^{-(\lambda-\lambda')y} \Big|_0^v \\
&= \frac{\lambda\lambda'}{\lambda-\lambda'} (e^{-\lambda' v} - e^{-\lambda v})
\end{aligned}$$

$$\begin{aligned}
G(v) &= \int_0^v g(t)dt \\
&= \int_0^v \frac{\lambda\lambda'}{\lambda-\lambda'}(e^{-\lambda' t}-e^{-\lambda t})dt \\
&= \frac{\lambda\lambda'}{\lambda-\lambda'}(\int_0^v e^{-\lambda' t}dt - \int_0^v e^{-\lambda t}dt) \\
&= \frac{\lambda\lambda'}{\lambda-\lambda'}(-\frac{1}{\lambda'}e^{-\lambda' t}|_0^v + \frac{1}{\lambda}e^{-\lambda t}|_0^v) \\
&= \frac{\lambda\lambda'}{\lambda-\lambda'}[\frac{1}{\lambda}(e^{-\lambda v}-1) - \frac{1}{\lambda'}(e^{-\lambda' v}-1)] \\
&= \frac{\lambda'}{\lambda-\lambda'}(e^{-\lambda v}-1) - \frac{\lambda}{\lambda-\lambda'}(e^{-\lambda' v}-1)
\end{aligned}$$

$$\begin{aligned}
f(u) &= \int_0^u \int_0^{u-y} f_X(x)f_Y(y)f_Z(u-x-y)dxdy \\
&= \int_0^u \int_0^{u-y} \lambda e^{-\lambda x}\lambda e^{-\lambda y}\lambda' e^{-\lambda'(u-x-y)}dxdy \\
&= \lambda^2\lambda' e^{-\lambda' u}\int_0^u e^{-(\lambda-\lambda')y} \\
&\quad \int_0^{u-y} e^{-(\lambda-\lambda')x}dxdy \\
&= -\frac{\lambda^2\lambda'}{\lambda-\lambda'}e^{-\lambda' u}\int_0^u [e^{-(\lambda-\lambda')(u-y)}-1]e^{-(\lambda-\lambda')y}dy \\
&= -\frac{\lambda^2\lambda'}{\lambda-\lambda'}e^{-\lambda' u}\int_0^u [e^{-(\lambda-\lambda')u}-e^{-(\lambda-\lambda')y}]dy \\
&= -\frac{\lambda^2\lambda'}{\lambda-\lambda'}e^{-\lambda' u}[ue^{-(\lambda-\lambda')u} - \int_0^u e^{-(\lambda-\lambda')y}dy] \\
&= \frac{\lambda^2\lambda'}{(\lambda-\lambda')^2}[e^{-\lambda' u}-(u(\lambda-\lambda')+1)e^{-\lambda u}]
\end{aligned}$$

$$\begin{aligned}
F(u) &= \int_0^u f(t)dt \\
&= \int_0^u \frac{\lambda^2\lambda'}{(\lambda-\lambda')^2}[e^{-\lambda' t}-(t(\lambda-\lambda')+1)e^{-\lambda t}]dt \\
&= \frac{\lambda^2\lambda'}{(\lambda-\lambda')^2}[\int_0^u e^{-\lambda' t}dt - \int_0^u t(\lambda-\lambda')e^{-\lambda t}dt \int_0^u e^{-\lambda t}dt] \\
&= \frac{\lambda^2\lambda'}{(\lambda-\lambda')^2}[-\frac{1}{\lambda'}e^{-\lambda' t}|_0^u + \frac{1}{\lambda}e^{-\lambda t}|_0^u - (\lambda-\lambda')\int_0^u te^{-\lambda t}dt]
\end{aligned}$$

$$
\begin{aligned}
&= \frac{\lambda^2\lambda'}{(\lambda-\lambda')^2}[\frac{1}{\lambda}e^{-\lambda u} - \frac{1}{\lambda'}e^{-\lambda' u} + \frac{1}{\lambda'} - \frac{1}{\lambda} \\
&\quad -(\lambda-\lambda')(-\frac{t}{\lambda}e^{-\lambda t}|_o^u - \int_0^u (-\frac{1}{\lambda})e^{-\lambda t}dt)] \\
&= \frac{\lambda^2\lambda'}{(\lambda-\lambda')^2}[\frac{1}{\lambda}e^{-\lambda u} - \frac{1}{\lambda'}e^{-\lambda' u} + \frac{1}{\lambda'} - \frac{1}{\lambda} \\
&\quad +\frac{\lambda-\lambda'}{\lambda}ue^{-\lambda u} - \frac{\lambda-\lambda'}{\lambda}(-\frac{1}{\lambda})e^{-\lambda t}|_0^u] \\
&= \frac{\lambda^2\lambda'}{(\lambda-\lambda')^2}[\frac{1}{\lambda}e^{-\lambda u} - \frac{1}{\lambda'}e^{-\lambda' u} + \frac{1}{\lambda'} - \frac{1}{\lambda} \\
&\quad +\frac{\lambda-\lambda'}{\lambda}ue^{-\lambda u} + \frac{\lambda-\lambda'}{\lambda^2}(e^{-\lambda u} - 1)] \\
&= \frac{\lambda\lambda'}{(\lambda-\lambda')^2}e^{-\lambda u} - \frac{\lambda^2}{(\lambda-\lambda')^2}e^{-\lambda' u} + \frac{\lambda^2}{(\lambda-\lambda')^2} - \frac{\lambda\lambda'}{(\lambda-\lambda')^2} \\
&\quad +\frac{\lambda\lambda'}{\lambda-\lambda'}ue^{-\lambda u} + \frac{\lambda'}{\lambda-\lambda'}e^{-\lambda u} - \frac{\lambda'}{\lambda-\lambda'}
\end{aligned}
$$

$$
F_T(t) = G(t)F(t)^{m-1}
$$

$$
\begin{aligned}
f_T(t) &= \frac{\partial F_T(t)}{\partial t} \\
&= \frac{\partial G(t)F(t)^{m-1}}{\partial t} \\
&= (m-1)F(t)^{m-2}\frac{\partial F(t)}{\partial t}G(t) + F(t)^{m-1}\frac{\partial G(t)}{\partial t} \\
&= (m-1)f(t)F(t)^{m-2}G(t) + g(t)F(t)^{m-1}
\end{aligned}
$$

References

[1] Johnathan Bein, Roger King & Nabil Kamel, "MOBY: An Architecture for Distributed Expert Database Systems," *Proc. of the 13th Int'l Conf. on VLDB*, 1987, 13–20.

[2] D. Bitton, H. Boral, D. J. DeWitt & W. K. Wilkinson, "Parallel Algorithms for the Execution of Relational Database Operations," *ACM Transactions on Database Systems*, Vol. 8, No. 3, September 1983, 324–353.

[3] K. Mani Chandy & Leslie Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Transactions on Computer Systems*, Vol. 3, No. 1, February 1986, 64–75.

[4] Simona Rabinovichi Cohen & Ouri Wolfson, "Why a Single Parallelization Strategy is not Enough in Knowledge Bases," *Principle of Database Systems (PODS)*, 1989, 200–216.

[5] U. Dayal, et al., "The HiPAC Project: Combining Active Databases and Timing Constraints," *SIGMOD Record*, Vol. 17, No. 1, March 1988, 51–70.

[6] Guozhu Dong, "On Distributed Processibility of Datalog Queries by Decomposing Databases," *SIGMOD Record*, Vol. 18, No. 2, 1989, 26–35.

[7] The Committee for Advanced DBMS Functions, "Third-Generation Database System Manifesto," *SIGMOD Record*, Vol. 19, No. 3, September 1990, 31–44.

[8] Sumit Ganguly, Avi Silberschatz & Shalom Tsur, "A Framework for the Parallel Processing of Datalog Queries," *SIGMOD Record*, Vol. 19, Iss. 2, 1990, 143–152.

[9] Anoop Gupta, *Parallelism in Production Systems*. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.

[10] Eric N. Hanson, "An Initial Report on the Design of Ariel," *SIGMOD Record*, Vol. 18, No. 3, September 1989, 12–19.

[11] Wilson Harvey, et al., "The Effectiveness of Task-Level Parallelism for Production Systems," *Journal of Parallel and Distributed Computing*, Vol. 13, No. 4, December 1991, 383–394.

[12] Guy Hulin, "Parallel Processing of Recursive Queries in Distributed Architectures," *Proc. of the 15th Int'l Conf. on VLDB*, Amsterdam, The Netherlands, 1989, 87–96.

[13] Toru Ishida, "Parallel Rule Firing in Production Systems," *IEEE Transactions on Knolwdge and Data Engineering*, Vol. 3, No. 1, March 1991, 11–17.

[14] Michael A. Kelly & Rudolph E. Seviora, "An Evaluation of DRete on CUPID for OPS5 Matching," *Proc. of the '89 Int'l Joint Conf. on Artificial Intelligence*, 1989, 84–90.

[15] Steve Kuo & Dan Moldovan, "Implementation of Multiple Rule Firing Production Systems on Hypercube," *Journal of Parallel and Distributed Computing*, Vol. 13, December 1991, 383–394.

[16] L. Lamport, "Time, Clocks and the Ordering the Events in a Distributed Systems," *Communication of the ACM*, Vol. 21, No. 7, July 1978, 558–565.

[17] Kemal Oflazer, "Partitioning in Parallel Processing of Production Systems," *Pro. of Int'l Conf. on Parallel Processing*, 1984, 92–100.

[18] A. O. Oshisanwo & P. P. Dasiewicz, "A Parallel Model and Architecture for Production Systems," *Int'l Conf. on Parallel Processing*, 1987, 147–153.

[19] Louiqa Raschid, Timos K. Sellis & A. Delis, "On the Concurrent Execution of Production Rules in a Database Implementation," *CS-TR-2751, Department of Computer and Information Science, University of Maryland*, 1991.

[20] James G. Schmolze, "Guaranteeing Serializable Results in Synchronous Parallel Production Systems," *Journal of Parallel and Distributed Computing*, Vol. 13, No. 4, December 1991, 348–365.

[21] James G. Schmolze & Suraj Goel, "A Parallel Asynchronous Distributed Production System," *Proc. of the '87 Int'l Conf. on Artificial Intelligence*, July 13-17, 1987, 65–71.

[22] Avi Silberschatz, Michael Stonebraker & Jeffrey D. Ullman, "Database Systems: Achievements and Opportunities," *SIGMOD Record*, Vol. 19, No. 4, December 1990, 6–22.

[23] M. Spezialetti & J. P. Kearns, "Simultaneous Regions: A Framework for the Consistent Monitoring of Distributed Systems," *Proc. of Int'l Conf. on Distributed Computing Systems*, 1989, 61–68.

[24] Jaideep Srivastava, Kuo-Wei Hwang & Jack S. Eddy Tan, "Parallelism in Database Production Systems," *Proc. of the 6th Int'l Conf. on Data Engineering* , 1990, 121–128.

[25] Salvatore J. Stolfo, Daniel Miranker & R. Mills, "A Simple Processing Scheme to Extract and Load Balance Implicit Parallelism in The Concurrenrt Match of Production Rules," *Proc. AFIPS Symposium on Fifth Generation Computing*, 1985.

[26] Salvatore J. Stolfo & Daniel P. Miranker, "The DADO Production System Machine," *Journal of Parallel and Distributed Computing*, Vol. 3, 1986, 269–296.

[27] Michael Stonebraker, Eric N. Hanson & Spyros Potamianos, "The POSTGRES Rule Manager," *IEEE Transactions on Software Engineering*, Vol. 14, No. 7, July 1988, 897–907.

[28] Michael Stonebraker, Marti Hearst & Spyros Potamianos, "A Commentary on the POSTGRES Rules System," *SIGMOD Record*, Vol. 18, No. 3, September 1989, 5–11.

[29] Jeffrey D. Ullman, *Principles of Database Systems*. Computer Science Press, Rockville, Maryland, 1982.

[30] Ouri Wolfson & Aya Ozeri, "A New Paradigm for Parallel and Distributed Rule-Processing," *SIGMOD Record*, Vol. 19, Iss. 2, 1990, 133–142.