Inform. Systems Vol. 9, No. 2, pp. 121–138, 1984
Printed in the U.S.A.

0306–4379/84 $3.00 + .00
© 1984 Pergamon Press Ltd.

SEMANTIC INTEGRITY ENFORCEMENT IN CENTRALIZED DBMS AND DISTRIBUTED DBMS BASED ON SLOW (GEOGRAPHICALLY DISTRIBUTED) AND FAST (LOCAL AREA) NETWORKS

DUSHAN Z. BADAL
Computer Science Department, Naval Postgraduate School, Monterey, CA 93940, U.S.A.

(*Received* 31 *January* 1983; *in revised form* 25 *July* 1983)

Abstract—In this paper we investigate techniques for decreasing the overhead of semantic integrity enforcement or equivalently the overhead of transaction validation with respect to a set of semantic integrity (SI) assertions. We discuss the problem of semantic integrity enforcement from two points of view. First we describe three approaches to decrease the overhead of SI enforcement. Second we analyze the cost of several SI enforcement methods in centralized and distributed database systems based on slow and fast (local) networks.

1. INTRODUCTION

Many applications require that the users of a database be able to specify semantic integrity (SI) assertions or constraints about their data. SI assertions determine semantic consistency or semantic integrity of data in the database, i.e. they delimit values in the database in terms of other database values or constants. In order to preserve database semantic integrity the transactions must be validated with respect to SI assertions and all transactions which generate database states for which SI assertions would evaluate to false must be rejected. We can say that SI assertions must be enforced in order to maintain database semantic integrity.

Although considerable work has been done on the specification methodology for SI assertions[15, 18, 22, 8, 14, 7, 16, 21, 4] only a few papers deal with SI enforcement implementation issues[19, 5, 20, 3, 10].

A major problem in validating transactions with respect to a set of SI assertions is the high overhead (or cost) caused by the dependency of transactions and SI assertions on values stored in the database. Such dependencies prevent *a priori* proofs of transaction correctness with respect to a set of SI assertions. An example of such data dependent transaction T and SI assertion A could be as follows:

T: increase the salary of employee J. Johnson by 10%

A: salary of employee < MAX (salary of manager, 1.5*average salary)

The transactions whose SI correctness cannot be proven must be monitored whether or not their outputs violate SI assertions. The subsystem which monitors such SI assertion violations is properly a part of the database management system and in this paper we investigate several methods of SI enforcement or SI validation. First we suggest several heuristic techniques for SI enforcement. Such techniques are intended to decrease the overhead associated with SI enforcement. We consider these techniques independently of transaction execution and transaction validation sequencing, i.e. we first investigate how, rather than when, to effectively enforce SI. In the second part of the paper we investigate the cost of SI enforcement as a function of sequencing and interleaving of transaction execution and transaction validation processes. Thus, the second part of the paper is primarily concerned with the problem of when, rather than how, to enforce SI.

1.1 *Survey of published SI enforcement methods*

Four SI enforcement methods have been reported in the literature[18, 19, 5, 10, 3]. Stonebraker[18, 19] suggested and partially implemented SI enforcement by a query modification method when the query qualification condition is modified so that it contains SI assertions related to the query.

The SI enforcement proposed in System R[5] is based on a trigger concept which is close to "on condition" or exception handling in programming languages. The trigger is a block of statements consisting of a body and a condition. The trigger is executed whenever the condition becomes true. The trigger condition is the SI assertion with the specification of when it should hold, i.e. depending on DB state, DB transition or the type of statement(s). The trigger body is an algorithm for the corrective action(s) in case the SI assertion has not been complied with, i.e. the triggering event occurred.

The principal idea of the SI validation method described in[10] is to analyze the effect of each database operation on each SI assertion. Such analysis is logical in nature and can be performed at database design time. Clearly, such approach applies only to *a priori* known (or predefined) database operations and SI assertions. The result of such analysis, called perturbation analysis, is the decision by an assertion processor as to whether a given transaction can be validated before or during its execution and the generation of SI tests for such validation. If several candidate SI tests can be derived, then the most efficient one is used in SI enforcement. SI test can be executed before the transaction execution (prompt SI tests) or during the transaction execution. It is suggested in[10] that such SI enforcement is efficient for the following reasons:

(1) The SI tests generated by the assertion processor are more efficient than the evaluation of SI asertions.

(2) The method avoids transaction rollback.

The principal ideas of SI enforcement described in[3] can be summarized as follows. If the SI assertion has arguments distributed over several relations then SI enforcement can be done in two overhead-saving ways:

(1) Check after each step of SI assertion evaluation whether the transaction which has triggered assertion evaluation is a Readily Ignorable Update (RIU). This means that the SI assertion is always evaluated in the same order given by an assertion evaluation tree. Each node of such evaluation tree contains a relevance vector and a difference vector which together determine whether a given update for each level of partial evaluation of SI assertion is or is not an RIU. If a given update is found to be an RIU then SI assertion partial evaluation can be terminated as the update has been found irrelevant with respect to a given SI assertion.

(2) Execute the transaction and evaluate SI assertions completely but only on a subset of the data. Buneman and Clemons[3] call this technique filters. If an update does not pass the filter then the SI assertion cannot evaluate to true even on a full range of data. However, if the update does pass the filter, this merely indicates that SI assertion may evaluate to true and thus transaction execution and SI validation must be performed on a full range of data.

2. STRATEGIES TO DECREASE SI ENFORCEMENT OVERHEAD

In this section we outline three additional strategies for SI enforcement and we point out in which way they can decrease SI enforcement overhead.

2.1 *Cost incremental SI enforcement*

If the transaction is to be validated with respect to a set of SI assertions which can be evaluated at different costs, then the transaction validation should be in a cost incremental manner. This means that the transaction should first be validated with respect to "least expensive" SI assertions and last with respect to "most expensive" SI assertions. SI enforcement cost can be decreased because if the transaction violates less costly assertions one avoids evaluation of "expensive" assertions.

In order to realize the cost incremental SI enforcement it is necessary to analyze the cost of each SI assertion evaluation. One valid cost criterion is the number of secondary storage accesses needed for SI assertion evaluation. In distributed databases a communication cost is another valid and frequently used cost criterion. We suggest here two additional higher level cost criteria. They are the type of mapping represented by an SI assertion and the direction of SI assertion evaluation. They both can be determined from a logical schema because they describe the semantics of the application modelled by database.

We assume that SI assertions can represent three kinds of mappings:

(1) one-to-one

(2) one-to-many

(3) many-to-many.

For ease of exposition we assume the syntax of an SI assertion to be an expression followed by an operator and another expression i.e. expr 1 op expr 2. In one-to-one SI assertions both expressions are single valued. For example, both can represent one attribute value of one tuple in different relations.

In one-to-many SI assertions expr 1 can be, a set of elements e.g. set of tuples in some relation. Expr 2 is a single value. Thus expr 2 can be a constant, a function, or an attribute value in some tuple.

In many-to-many SI assertions expr 1 and expr 2 are multiple valued, e.g. sets of tuples satisfying some qualification. The following examples illustrate the above mappings:

(1) one-to-one: salary of J. Jones in department $D1 >$ salary of J. Smith in department $D2$

(2) one-to-many: salary of any employee in department $D1 <$ salary of department $D1$ manager

(3) many-to-many: salary of any employee in department $D1$ $=/=$ salary of any employee in department $D2$

Many SI assertions represent one of the above mappings regardless of their syntax. SI assertions of type one-to-one are the least expensive to evaluate, and the SI assertions of type many-to-many are the most expensive to evaluate because of the number of database accesses and computing time. It is important to realize that any SI assertion can be evaluated in two directions. The direction (i.e. the order) of SI assertion evaluation (expr 1 expr 2 or expr 2 expr 1) depends on the name of the variable(s) being modified by the transaction and the position(s) of the variable(s) in the SI assertion. In one-to-one and many-to-many SI assertions the cost of SI assertion evaluation is independent of the direction of SI assertion evaluation. However, in a many-to-one SI assertion the cost of SI enforcement might depend on the direction of SI assertions evaluation as follows.

Suppose that the transaction $T1$ updated the salary of several employees in department $D1$ and transaction $T2$ updated the salary of the department $D1$ manager. Clearly, the cost of validating $T1$ and $T2$ with respect to SI assertion $A1$: "salary of any employee in department $D1 <$ salary of the department $D1$ manager," will be quite different. The reason being that the transactions $T1$ and $T2$ require the SI assertion evaluation in opposite directions, i.e.

$T1$ requires evaluation of $A1$ in the direction many-to-one, and $T2$ requires the evaluation of $A1$ in the direction one-to-many. In terms of secondary memory accesses and processing time it is less costly to compare the new salary of several employees to the salary of the manager than to compare the new salary of the manager to the salary of every employee in the department $D1$. However it should be noted here that the direction of SI evaluation in general is dependent on the transaction or the database operation and it cannot be chosen by the user.

Considering only the types of mappings represented by SI assertions and the direction of their evaluation the cost incremental SI enforcement would require the evaluation of SI assertions, associated with or triggered by one transaction, in the following order:

(1) All one-to-one SI assertions.

(2) All one-to-many SI assertions which require evaluation in many-to-one direction.

(3) All one-to-many SI assertions which require evaluation in one-to-many directions.

(4) All many-to-many assertions.

It should be noted here that the above ordering of assertion evaluation is due to considering the types of mapping and the direction of evaluation only. This means that in general some other criteria, such as probabilities for the violation of some assertions, can affect the order of assertions evaluation.

2.2 *SI enforcement strategy using SI data*

SI data is a small portion of database data used frequently in SI validation and containing also the values of some aggregate functions which are the arguments of SI assertions. SI data is kept in a memory with faster access than the secondary memory used for database. It seems only natural to update the final value of, rather than to recompute, the aggregate function when one or more of its arguments are updated. The class of aggregate functions which can profitably use the concept of SI data should contain linear functions.

As an example, consider the costs involved in recomputing the aggregate function "average salary". If the "average salary" is a part of SI data then it is continually updated on any deletion, addition or update of employee's file. Another way to compute the "average salary" involves searching the whole employee file. It seems that under the most circumstances it should be less costly to use the former method, i.e. to use SI data.

2.3 *Conditional SI assertion evaluation*

Any SI assertion can have associated with it a condition describing the circumstances under which a transaction to be validated with respect to the SI assertion cannot violate such an assertion. These conditions must be derived for each SI assertion as they depend entirely on the semantics of each assertion. The derivation of these conditions amounts to logical analysis of SI assertion argument values which cannot lead to the violation of SI assertion. As an example, consider derivation of the condition for SI assertion SIAO: (expression 1) < (expression 2). The SI assertion SIAO can evaluate to false only if the value of expression 1 increases or the value of expression 2 decreases. Therefore, the conditions for SIAO are:

(C1) If transaction interacts only with expression 1 and decreases its value;

(C2) If transaction interacts only with expression 2 and increases its value;

(C3) If transaction increases the value of expression 1 by less than it increases the value of expression 2;

(C4) If transaction increases or decreases the values of expression 1 and expression 2 by the same amount;

(C5) If transaction decreases the value of expression 2 by less than it decreases the value of expression 1.

If condition C1 OR C2 OR C3 OR C4 OR C5 is true for some transaction T, then T cannot violate SI assertion SIAO, and therefore, T need not be validated with respect to SIAO.

3. ANALYSIS OF SI OVERHEAD

3.1 *Transaction execution and integrity enforcement models*

In the remaining part of the paper we investigate the sequencing and interleaving of transaction execution and SI enforcement processes and their impact on the cost of SI enforcement.

In order to investigate possible sequencings of transaction execution and SI enforcement processes we must model both. We model transaction execution in centralized DBS and at any site in distributed DBS as a two-phase process. The first phase of transaction execution consists of reading, computation of updates and perhaps writing the updates somewhere, but not into the original position, in the secondary storage. This could happen when a shadow page mechanism[11, 13, 17] is used or if capacity of the main storage is not sufficient to hold the updates. We call this phase EXEC. The second phase of transaction execution consists of writing the updates into the original position in a secondary storage. We call it UPDATE.

Our transaction execution model is intended to accommodate a variety of recovery strategies. It seems that there are two basic schools of thought as to how and when the update or better said the updated page should be written into the secondary storage. Both strategies aim at the same goal of writing the page into the original position in a secondary non-volatile storage. One strategy[11, 13, 17] is to write the updated page to a side file or shadow page and eventually to write the shadow page into the original page position. The second strategy is to write the updated page directly into the original page space[9, 12] and to support the recovery by writing a log into the non-volatile storage ahead of writing the updated page into the original page space.

The updated page can be written either when transaction commits or at some later time (which provides greater flexibility in choosing page replacement strategy). However, in both strategies the updated pages are eventually written in the original page space. Our transaction execution model and our cost analysis consider only the fact that the updates will be eventually written into the secondary non-volatile memory. This is the UPDATE phase in our transaction execution model. Thus our model does not reflect writes of updates to the secondary storage which might occur during the EXEC phase because such accesses are system configuration (e.g. primary memory capacity), transaction processing strategy and recovery mechanism dependent, and therefore are not included in a general cost analysis presented in this paper.

The only extension of transaction execution and SI enforcement models due to a distributed environment is that the UPDATE phase at each site of a distributed DBS becomes a part of a two phase commit protocol which implements or enforces the transaction atomicity in distributed environment.

Our SI enforcement model is intended to be able to represent all published SI enforcement methods. Thus we have separated the SI enforcement into two processes. One is the SI enforcement on a full set of data required by SI assertions and it is divided into two phases—SIREAD and SIEVAL. The second process is the SI enforcement on a subset of data as described in[3, 10] and we call it SI test. The SI test in reality would consist of SIREAD and SIEVAL phases executed on a small subset of data. However we model it as a one phase process in order to keep our analysis tractable.

As mentioned before our SI enforcement or transaction validation model consists of two phases: reading of SI assertion argument values which are not generated by the executing transaction (SIREAD) and evaluation of SI assertions (SIEVAL). However, the SIEVAL phase reads the data generated by the executing transaction and thus our model requires that SIEVAL phase occurs only after the transaction execution phase EXEC, although the SIREAD and the transaction EXEC phases can occur simultaneously. Our motivation for this model of SI evaluation is that in order to evaluate the integrity assertion(s) we have to obtain first its argument values from transaction output and possibly from other data in the database. We feel the requirement of SIEVAL phase following SIREAD phase is a realistic restriction for database or transaction systems requiring relatively few semantic integrity assertion evaluations. Most of the present systems seems to be of this sort. However for systems with extensive degree of integrity checking, i.e. when each transaction has a large set of integrity assertions associated with it, then our model of integrity enforcement may not be precise enough as SIREAD and SIEVAL phases could, and perhaps should, occur simultaneously and in an interleaved manner.

In this paper we investigate three SI enforcements. In pre-execution time SI enforcement the transaction is first validated and only then executed. In run-time SI enforcement the transaction execution and its validation are interleaved. In pre-execution-run time method the transaction validation is also interleaved with transaction execution. We do not consider the sequencing, investigated in[1, 2], in which the transaction is first executed, then the updated pages are written back to their original positions in the secondary storage and only then SI validation occurs. This sequencing, called post-execution SI validation, has been shown to have the highest cost.

3.2 *Pre-execution time SI enforcement*

Pre-execution time SI enforcement which is not the SI enforcement at a compile time, involves a subset of data and it occurs before the execution of transaction on a full set of data. Such limited—width transaction execution and SI assertions evaluation is called prompt SI test[10]. The use of such an SI test enables detection of SI violations that would result from the transaction execution before the database is modified. A somewhat related SI enforcement method has been proposed in[3]. From the proposals published in[3, 10] it is not entirely clear whether these SI validation methods are generally applicable. However, it is not the purpose of this paper to elaborate on that. We only want to model and evaluate the cost of published SI enforcement methods.

SI validation before transaction execution has one obvious advantage—it does not require transaction rollback when SI assertions are violated. However, it has two disadvantages. First, the method provides no concurrency between SI enforcement and transaction execution processes. Second, the database objects on which compile time SI tests are run cannot be modified by any other transaction until the one being validated is executed. Effectively, such database data objects must remain write-locked from SI validation through transaction execution, since SI tests must execute on the same database data values as the transaction will during its execution. Otherwise revalidation is required. Sequencing of events in compile time SI enforcement is shown in Fig. 1, where *n* is the total number of transactions, *m* is the number of transactions accepted, and *n*–*m* is the number of transactions rejected because of SI violations.

3.3 *Run time SI enforcement*

Run time SI enforcement means that SI enforcement and transaction execution are interleaved as shown in Fig. 2. EXEC and SIREAD phases can, if possible, occur concurrently. They are followed by SIEVAL phase and the UPDATE phase. The major advantage of run time SI enforcement is concurrent execution and interleaving of processes. Other advantages are: the time interval during which the database data must be locked for SI enforcement is reduced to transaction execution time, and duplicate reads dur-

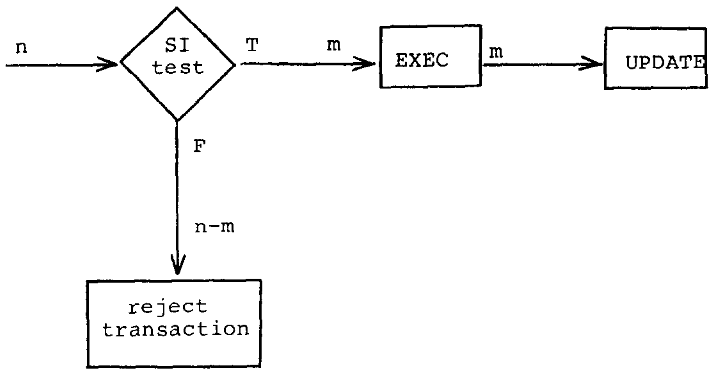

Fig. 1.

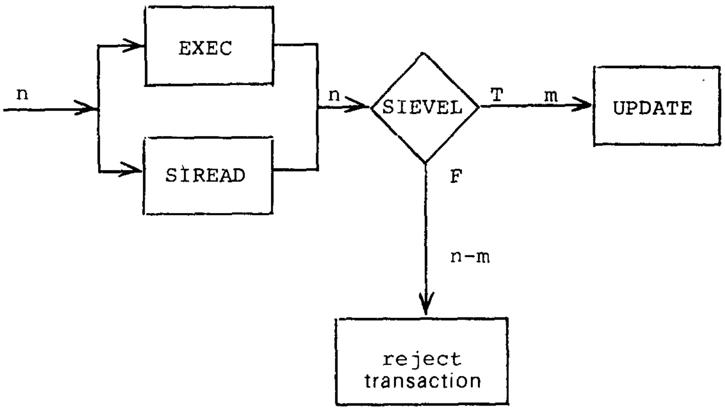

Fig. 2.

ing EXEC and SIREAD phases are avoided. Examples of run time SI enforcement can be found in [4, 18, 19].

3.4 *Pre-execution-run time SI enforcement*

Pre-execution-run time SI enforcement means that an SI test cannot decide whether the transaction will or will not violate SI assertions. This method was described in [3] and it was called a filter. The principal idea underlying pre-execution-run time SI enforcement is that if a filter or in our terminology SI test evaluates to true then this merely indicates that the transaction may not violate SI assertions. Thus, transaction revalidation is required after EXEC phase. However, if an SI test evaluates to false then the transaction will violate SI assertions and thus can be rejected without execution. The algorithm for this SI enforcement, called CR2, is shown in Fig. 3. We will also consider SI enforcement which is a logical complement of CR2. The algorithm for such pre-execution-run time SI enforcement, called CR1, is shown in Fig. 4.

The pre-execution-run time SI enforcement methods CR1 and CR2 have the same advantages as run time SI enforcement, e.g. avoidance of duplicate reads during SIREAD and EXEC. On the other hand, data objects used for SI enforcement must stay write-locked longer than in run time SI enforcement.

Our motivation for considering CR1 and CR2 SI enforcement methods is to investigate the overhead of both because depending on the application it may be easier to devise one SI test than the other.

3.5 *Centralized DBS*

The cost of SI enforcement in centralized DBS consists of two factors:

(1) Accessing database data in order to evaluate SI assertions.

(2) Computation to evaluate SI assertions.

We do not consider here the computational cost for SI assertion evaluation because we assume it to be the same for all SI enforcement methods. Moreover in present database systems the computational cost is a small fraction of the secondary memory accesses cost.

3.6 *Cost of SI enforcement in centralized DBS*

We derive the cost of each SI enforcement method in the context of transaction execution cost because each SI enforcement method requires a different number of transactions to be executed. The transaction execution cost and SI evaluation cost in centralized DBS is expressed in terms of the secondary

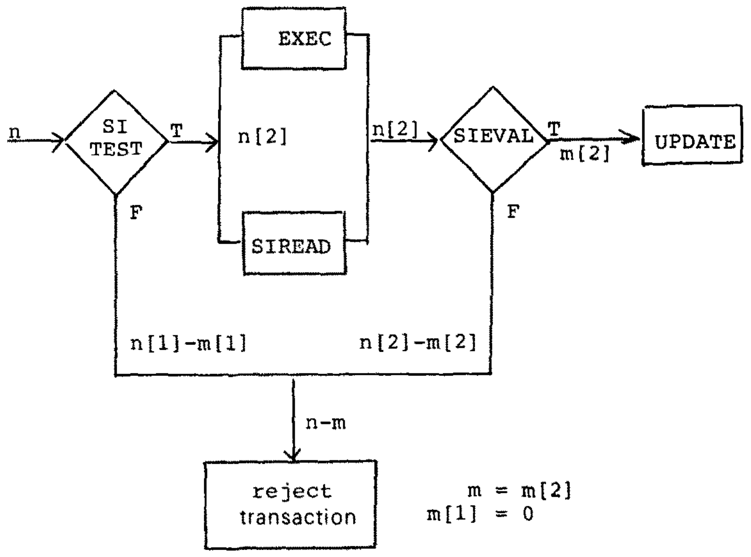

Fig. 3.

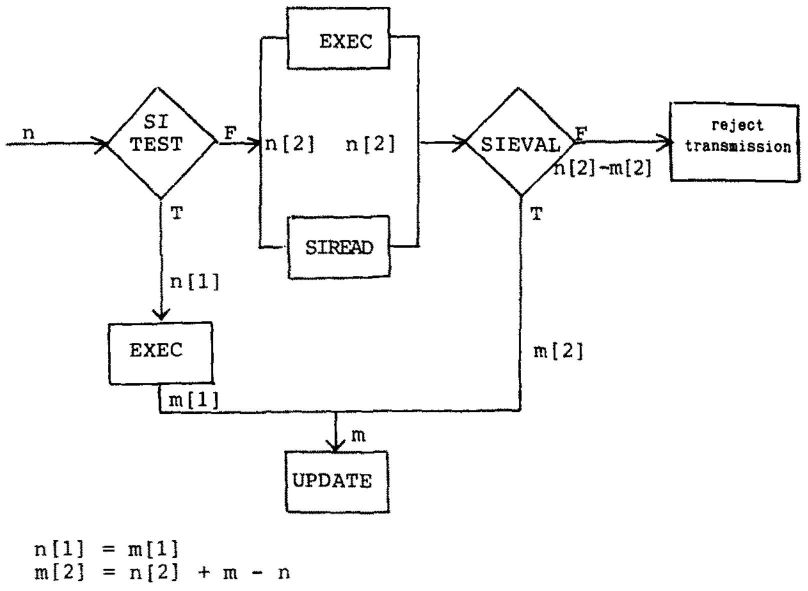

Fig. 4.

storage accesses. The cost derivations below have the following meaning—they represent the total cost of n transaction processing under the various SI enforcement methods.

Let X be the average number of read accesses made by a transaction; Y be the average number of write accesses made by a transaction; Z be the average number of those read accesses counted in X above that are also needed in transaction validation, i.e. Z is a subset of X which is needed for that part of transaction execution which enables evaluation of SI assertions. Clearly $Z \leqslant X$. W be the average number of accesses of data objects not accessed by the transactions but which are needed for SI validation. k expresses the efficiency of the SI test, i.e. it is the number equal to the ratio of cardinalities of a set of database data used by an SI test and a set of database data used for SI enforcement when no SI test is used. Clearly $0 < k \leqslant 1$.

The cost of pre-execution time SI enforcement can be derived using Fig. 1 and the above introduced numbers X, Y, Z, W and k as follows. Since under pre-execution time SI enforcement n transactions must be validated then one component of pre-execution time SI enforcement cost is:

$$kn(Z + W).$$

Since from n transactions submitted for execution under the pre-execution time SI enforcement only m

($m < n$) transactions will be accepted, i.e. $n-m$ transactions are rejected because they violated SI assertions, the cost of executing those m transactions under the pre-execution time SI enforcement is:

$$m(X - kZ + Y)$$

where we assume that kZ data accesses which occurred during the SI test will not be repeated during the transaction EXEC phase. Thus, the total cost of the pre-execution time SI enforcement (i.e. the cost of n transactions processing under pre-execution time SI enforcement) is:

$$C[ct] = kn(Z + W) + m(X - kZ + Y). \quad (1)$$

The cost of run time SI enforcement can be derived using Fig. 2 and X, Y, Z, W, k numbers as follows:

$$C[rt] = n(X + W) + mY. \quad (2)$$

The cost of CR1 pre-execution-run time SI enforcement can be derived using Fig. 4 and X, Y, Z, W, k numbers as follows:

$$\begin{aligned} C[cr1] = {} & kn(Z + W) + n[1](X - kZ + Y) \\ & + n[2](X - kZ) + (1 - k)n[2]W \\ & + m[2]Y. \quad (2a) \end{aligned}$$

Since in CR1 SI enforcement $n[1] = m[1]$ and $n = n[1] + n[2]$ then (2a) leads to:

$$\begin{aligned} C[cr1] = {} & (Z + W)(kn + n[2](1 - k)) + n(X - kZ) \\ & + Ym - (1 - k)n[2]Z. \quad (3) \end{aligned}$$

The cost of CR2 pre-execution-run time SI enforcement can be derived using Fig. 3 and X, Y, Z, W, k numbers as follows:

$$\begin{aligned} C[cr2] = {} & kn(Z + W) + n[2](X - kZ) \\ & + (1 - k)n[2]W + m[2]Y \\ = {} & (Z + W)(n[2](1 - k) + kn) + n[2](X - kZ) \\ & + mY - (1 - k)n[2]Z. \quad (4) \end{aligned}$$

3.6.1 *Cost comparisons.* Now that a consistent, straightforward method of expressing the cost of the various SI enforcement methods has been provided, it is useful to compare them. First we compare the costs assuming $k = 1$. Then from (2) and (3) we see that

$$C[rt] = C[cr1].$$

Assertion 1. The cost of CR2 pre-execution-run time SI enforcement when $k = 1$ is always lower than the cost of CR1 pre-execution-run time SI enforcement, i.e. $C[cr1] \geqslant C[cr2]$.

Proof

To prove that

$$C[cr1] \geqslant C[cr2]$$

we substitute from (3) and (4) and obtain

$$n \geqslant n[2] \quad (5)$$

which is always true for CR1 and CR2 SI enforcement.

Assertion 2. The cost of pre-execution time SI enforcement when $k = 1$ is always lower than the cost of CR1 pre-execution—run time SI enforcement, i.e. $C[ct] \leqslant C[cr1]$.

Proof

To prove that

$$C[cr1] \geqslant c[ct]$$

we substitute from (3) and (1) and obtain

$$n \geqslant m \quad (6)$$

which is always true.

Assertion 3. The cost of pre-execution time SI enforcement when $k = 1$ is always lower than the cost of CR2 pre-execution-run time SI enforcement, i.e. $C[ct] \leqslant C[cr2]$.

Proof

We prove $C[ct] \leqslant C[cr2]$ by contradiction. Suppose that the following is true:

$$C[ct] > C[cr2]. \quad (7)$$

By substituting into (7) from (1) and (4) we obtain

$$m \geqslant n[2]. \quad (8)$$

Since in the run time SI enforcement $m = m[2] = n[2]$ and in the CR2 pre-execution-run time SI enforcement $m[2] = m$ and $n[2] \geqslant m$ then $m \geqslant n[2]$ in (8) is impossible. Therefore $C[ct] \leqslant C[cr2]$ is true.

It is interesting to note that when $k = 1$ and when $Z = X$, i.e. when all database data read by the transaction are also used in its validation, then all SI enforcement methods have the same cost as follows:

$$C[ct] = C[ct] = C[cr1] = C[cr2] = n(X + W) + mY. \quad (8a)$$

However, if $Z \neq X$ and $k = 1$ then pre-execution time SI enforcement is the least costly and run time SI enforcement is the most costly, i.e.

$$C[rt] = C[cr1] > C[cr2] > C[ct] \quad (8b)$$

(8b) is important because even if SI tests are inefficient (i.e. $k = 1$), pre-execution time SI enforcement still has the lowest cost.

In order to establish a practical significance of (8b) we show the cost differences among different SI enforcement methods. We have chosen the following assumptions $X = 5$, $Y = 3$, $m = 95$, $n = 100$, $n[2] = 97$, $Z = W$ and $k = 1$.

By substituting the assumed values into (1), (2), (3) and (4) we obtain the cost of SI validation methods as shown in Fig. 5.

As can be seen from Fig. 5 the cost differences among SI enforcement methods when $k = 1$ are of the order of 2%.

We will now compare the cost of SI enforcement methods when SI tests are efficient, i.e. when $k < 1$.

Assertion 4. The cost of CR1 pre-execution-run time SI enforcement when $k < 1$ is always lower than the cost of run time SI enforcement, i.e. $C[cr] \leqslant C[rt]$.

Proof

To prove that

$$C[cr] \leqslant C[rt]$$

we substitute from (2) and (3) and obtain

$$n[2] \leqslant n \tag{9}$$

which is always true.

Assertion 5. The cost of pre-execution time SI enforcement when $k < 1$ is always lower than the cost of CR2 pre-execution-run time SI enforcement, i.e. $C[ct] \leqslant C[cr2]$.

Proof

To prove that

$$C[ct] \leqslant C[cr2]$$

we substitute from (1) and (4) and obtain

$$n[2]\left[1 + \frac{(1-k)W}{(X-kZ)}\right] \geqslant m \tag{10}$$

which is always true for CR2 pre-execution-run time and pre-execution time SI enforcement methods as in CR2 $n[2] > m$ and in pre-execution time $n[2] = n$.

Assertion 6. The cost of CR2 pre-execution-run time SI enforcement when $k < 1$ is always lower than the cost of CR1 pre-execution-run time SI enforcement, i.e. $C[cr2] \leqslant C[cr1]$.

Proof

To prove that

$$C[cr2] \leqslant C[cr1]$$

we substitute from (3) and (4) and obtain

$$n[2] \leqslant n \tag{11}$$

which is always true.

Thus, we have shown that in centralized DBS when $k < 1$ the following holds:

$$C[ct] \leqslant C[cr2] \leqslant C[cr1] \leqslant C[rt].$$

In order to show the cost differences among SI enforcement methods we analyze three different cases. We use the same parameter setting in all three cases, i.e. $X = 5$, $Y = 3$, $m = 95m$, $n[2] = 97$, $n = 100$, $n[1] = 3$. In scenario 2 we assume medium level of SI enforcement by setting $Z = W = 2$. In scenario 1 and scenario 3 we assume low ($Z = W = 0.1$) and high ($Z = W = 4$) levels of SI enforcement, respectively. Within each scenario we compute $C[rt]$, $C[ct]$, $C[cr1]$ and $C[cr2]$ as functions of SI test efficiency k. The results are shown for scenario 1 in Fig. 6, for scenario 2 in Fig. 7 and for scenario 3 in Fig. 8.

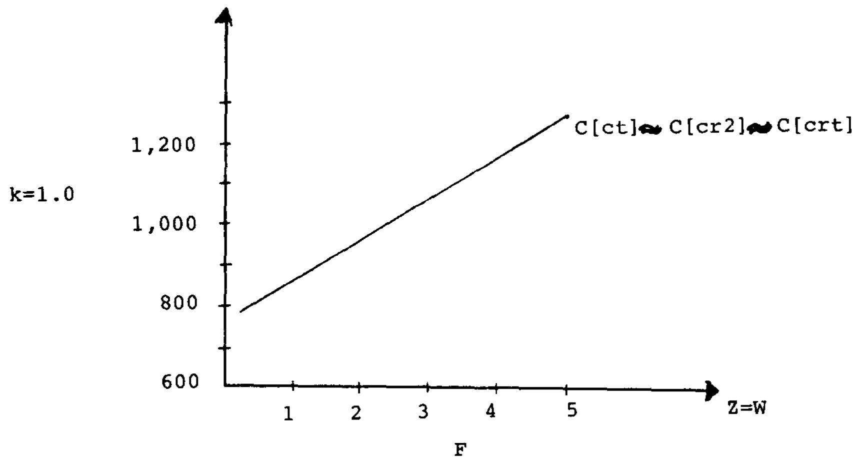

Fig. 5.

The next three figures show $C[rt]$, $C[ct]$, $C[cr1]$ and $C[cr2]$ as functions of SI validation level ($Z = W \in \langle 0.1, 5 \rangle$) for three SI test efficiencies $k = 0.1$ (Fig. 9), $k = 0.5$ (Fig. 10) and $k = 0.9$ (Fig. 11).

The results of our analysis show that there are very small cost differences among the run time, CR1 and CR2 pre-execution-run time SI enforcement methods. However, the pre-execution time SI enforcement can have significantly lower SI overhead than other SI enforcement methods. Also as expected the pre-execution time SI enforcement cost is decreasing proportionally to the increasing level of SI enforcement and increasing efficiency of SI tests.

3.7 *Distributed DBS*

In this section we derive the cost of different SI enforcement methods for distributed DBS based on slow and fast networks. A network is slow if its speed is substantially lower than that of secondary memory channels. Such slow networks are sometimes called geographically distributed networks. Thus, in the cost analysis of distributed DBS using slow network we neglect the cost of secondary memory accesses at each site and consider only the communication cost in terms of number of messages needed for transaction execution and its validation. As in the case of centralized DBS we consider the analysis which deals with

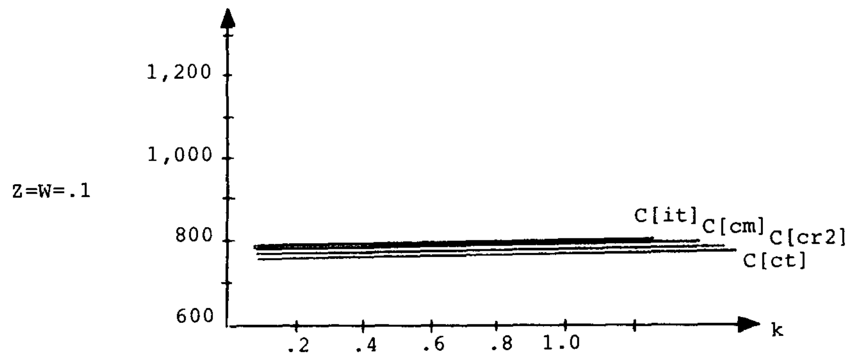

Fig. 6.

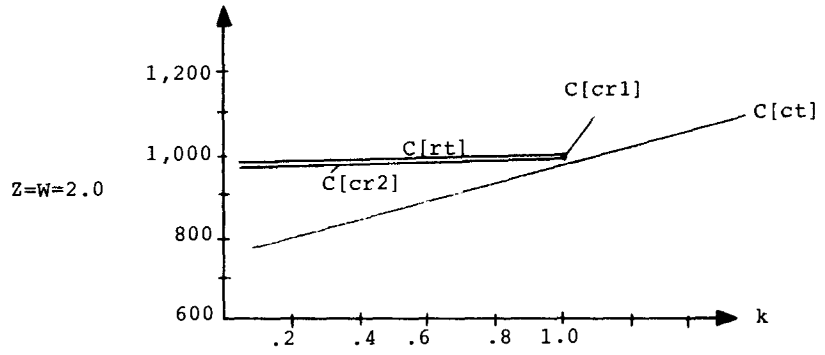

Fig. 7.

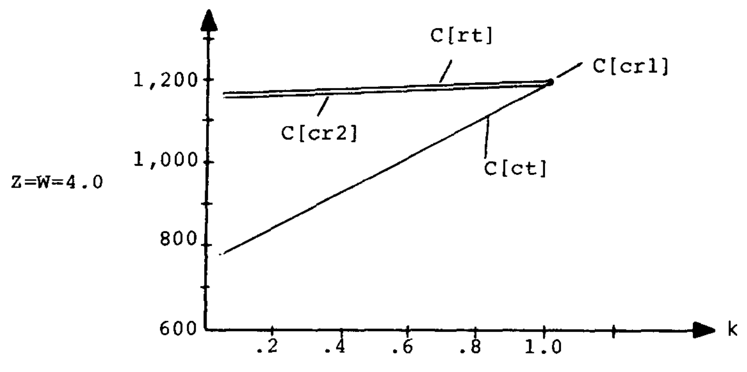

Fig. 8.

the total cost of transaction processing under the various SI enforcement methods. The reason for doing so is that different SI enforcement methods require (as indicated in Figs. 1–4) different number of transactions to be executed. However, in distributed DBS the transaction execution and its validation cost depends on the type of concurrency control mechanism under which transactions execute. Therefore our cost analysis reflects the communication cost needed for transaction synchronization. In this paper we consider centralized and distributed versions of two phase locking. We have chosen two phase locking as it seems to be the most widely used concurrency control mechanism at the present time. However, the same kind of analysis can be extended for an arbitrary concurrency control mechanism.

A network is fast if its speed is comparable to that of secondary memory channels. In many cases such

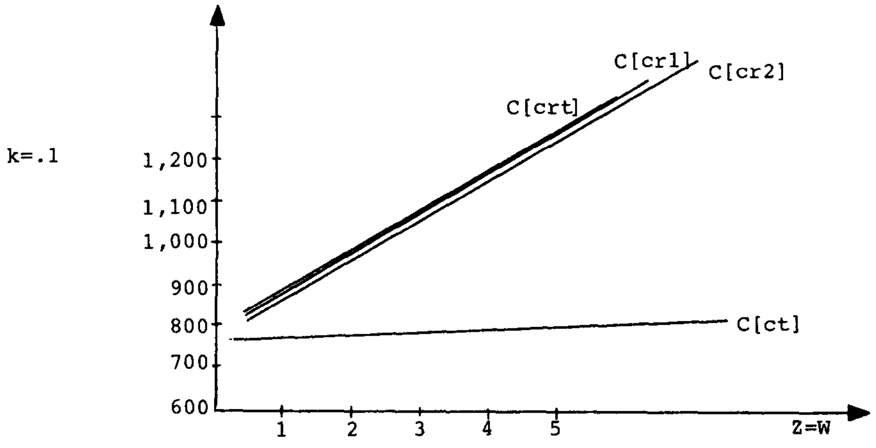

Fig. 9.

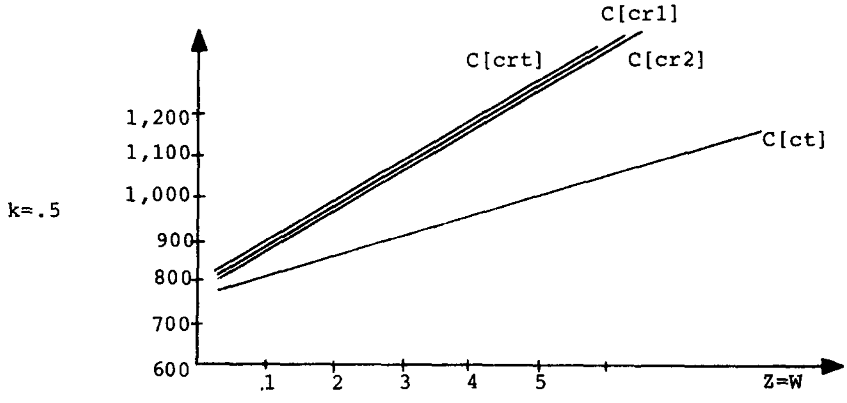

Fig. 10.

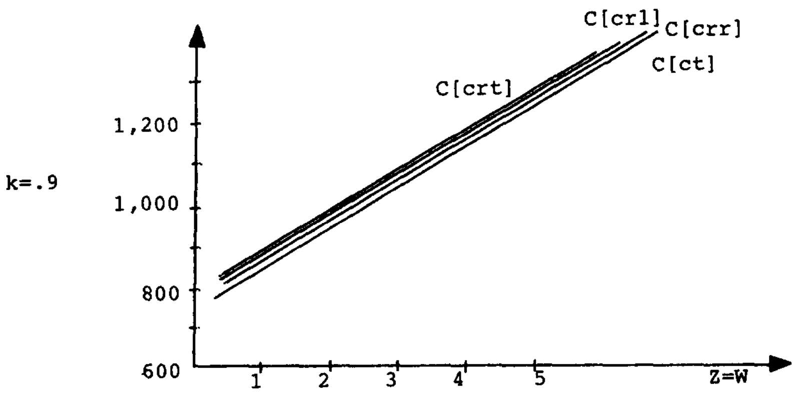

Fig. 11.

fast networks are called local networks. In the cost analysis of SI enforcement methods for fast network based distributed DBS we consider both the cost of secondary memory accesses at each site and the communication cost in terms of messages needed to synchronize, execute and validate the transaction. We again assume that distributed and centralized two phase locking is used as a concurrency control. Two phase locking (2PL) has been described in[6, 9, 12]. The basic difference between centralized and distributed 2PL is that in the former there is only one active system-wide or global lock table and thus all transactions must lock and unlock there. On the contrary in distributed 2PL there are lock tables local to each site and thus each transaction must lock and unlock at the sites where it has executed.

In our analysis we assume that the transaction atomicity is achieved by using a two-phase commit[11, 9, 12]. As mentioned in a preceding section we analyze only three types of SI enforcement methods—run time, pre-execution time and pre-execution-run time. The description of each method and its advantages and disadvantages is in Sections 3.2–3.4. The only extension to our transaction execution and SI enforcement models due to distributed environment is that the EXEC phase becomes a part of two-phase commit.

3.8 *Cost of SI enforcement in distributed DBS based on a slow network*

The following variables will be used for SI enforcement cost analysis in distributed DBS based on both slow and fast networks.

Let P be the average number of sites at which the transaction during its execution reads and writes or reads only; Q be the average number of sites at which the transaction during its execution writes only; S be the average number of sites counted in P above at which the transaction reads and which are also needed for SI enforcement. Clearly, $S \leqslant P.R$ be the average number of sites not accessed by the transaction but accessed for SI enforcement only. C be the SI test efficiency, i.e. it is the number equal to the ratio of the average number of sites accessed during SI test execution to the average number of sites accessed for SI enforcement when no SI test is used.

We derive the cost of each SI enforcement method by considering n transaction execution under the centralized and distributed two phase locking with two phase commit. We assume that from n transactions only m transactions are executed and n–m transactions are rejected because of SI violations. We use c–2PL & 2PC and d-2PL & 2PC to denote centralized and distributed two phase locking with two phase commit. We assume that in $c-2PL$ the data objects are all locked before transaction execution begins and that in d–2PL the data objects are being locked in an incremental manner, i.e. on demand during transaction execution. In deriving the cost of each SI enforcement method for distributed databases we first construct a SI enforcement algorithm for each SI enforcement method described in Sections 3.2–3.4 and Figs. 1–4. Then we compute the cost (i.e. the number of messages) for each algorithm.

Algorithm CTCS *for pre-execution time SI enforcement under* c–*2PL & 2PC*

(1) Acquire locks for $C(R+S)$ sites.

(2) Execute SI test, i.e. execute transaction and evaluate SI assertions at $C(R+S)$ sites, and either send the result to one site or send data to one site which then evaluates SI assertion.

(3) If SI test results in SI violation then reject transaction, release locks at $C(R+S)$ sites and terminate, else do 4.

(4) Acquire locks for $Q+P-C^*S$ sites and execute transaction EXEC phase.

(5) Execute UPDATE phase using two phase commit and release locks. In the first phase of 2PC the updates are sent to their sites and are acknowledged. In the second phase of 2PC the updates are committed, i.e. made available to other transactions and eventually written into the secondary non-volatile storage. Then the site which coordinated transaction execution releases all transaction locks at the central lock table and transaction execution and its validation terminates.

The number of messages generated at each step of the *CTCS* algorithm is:

(1) $2n$
(2) $2nC(R+S)$
(3) $2(n-m)$
(4) $2m$
(5) $2m+3mQ$.

The cost of CTCS algorithm is:

$$C[ctcs] = 4n + 2nC(R+S) + 3mQ. \qquad (12)$$

Algorithm RTCS *for run time SI enforcement under* c–*2PL & 2PC*

(1) Acquire locks for $(P+R)$ sites.

(2) Execute transaction EXEC phase and SIREAD phase.

(3) Execute SIEVAL phase

(4) If the result of 3 is false reject transaction, delete $(P+R)$ locks and terminate, else do 5.

(5) Execute transaction UPDATE phase using 2PC and release locks.

The number of messages generated by the *RTCS* algorithm at each step is:

(1) $2n$
(2) none
(3) $2n(R+S)$
(4) $2(n-m)$
(5) $2m+3mQ$.

The cost of *RTCS* algorithm is:

$$C[rtcs] = 4n + 2n(R+S) + 3mQ. \qquad (13)$$

From (12) and (13) by inspection we see that

$$C[rtcs] \geqslant C[ctcs].$$

In order to see the differences between $C[rtc]$ and $C[ctc]$ we compute

$$\frac{C[rtcs]}{C[ctcs]} = \frac{2n + (R + S) + 3mQ}{2n + C(R + S) + 3mQ} \tag{14}$$

Since in (14) $2n + 3mQ \gg C(R + S)$ then we can conclude that

$$C[rtcs] \approx C[ctcs] \tag{15}$$

Because of (14) we consider the analysis of CR1 and CR2 pre-execution-run time SI enforcement methods under c–2PL & 2PC unnecessary and we conclude that in the distributed DBS based on the slow network with centralized two phase locking and preclaiming of data the cost of all SI enforcement methods is essentially the same.

We will investigate now the cost of SI enforcement methods in distributed DBS based on the slow network when the concurrency control mechanism used is distributed two phase locking and the data objects needed by the transaction are locked on demand during the transaction execution.

Algorithm RTDS *for run time SI enforcement under* d–2*PL* & 2*PC*

(1) Lock at $(P + R)$ sites, execute EXEC phase and SIREAD phase.

(2) Evaluate SI assertions, i.e. send values of SI arguments from $(R + S)$ sites to one site for evaluation of SI assertions or $(R + S)$ sites send the results of SI assertions evaluation to one site.

(3) If the result of SI assertion evaluation is false then reject transaction, unlock $(P + R)$ sites and terminate, else do 4.

(4) Lock at Q sites, execute transaction UPDATE phase and release locks using 2PC, i.e. in the first phase of 2PC the "unlock" messages are broadcasted to $(P + R)$ sites and simultaneously Q "lock-update" messages are broadcasted to Q sites. After all $(P + R + Q)$ messages have been acknowledged then in the second phase of 2PC "execute update and execute unlock" and "execute unlock" messages are broadcasted to Q and $(P + R)$ sites respectively.

The number of messages generated by the *RTDS* algorithm at each step is:

(1) None (as there are no explicit lock messages)
(2) $2n(R + S)$
(3) $2(n - m)(P + R)$
(4) $3m(P + Q + R)$.

The cost of *RTDS* algorithm is

$$C[rtds] = 2n(R + S) + 2(n - m)(P + R) + 3m(P + Q + R). \tag{16}$$

Algorithm CTDS *for pre-execution time SI enforcement under* d–2*PL* & 2*PC*

(1) Lock at $C(R + S)$ sites and execute SI test.

(2) If SI test is false reject transaction, unlock $C(R + S)$ sites and terminate, else do 3.

(3) Execute transaction EXEC phase.

(4) Lock at Q sites, execute UPDATE phase and release locks using 2PC.

The number of messages generated by algorithm *CTDS* at each step is:

(1) $2nC(R + S)$
(2) $2C(n–m)(R + S)$
(3) None
(4) $3m(P + Q + R)$.

The cost of *CTDS* algorithm is:

$$C[ctds] = 2nC(R + S) + 2C(n - m)(R + S) + 3m(P + Q + R). \tag{17}$$

Algorithm CR1DS *for* CR1 *pre-execution-run time SI enforcement under* d–2*PL* & 2*PC*

(1) Lock at $C(R + S)$ sites and execute SI test.

(2) If the result of SI test is true execute transaction EXEC phase, lock at Q sites and execute transaction UPDATE phase and release locks using 2PC, else do 3.

(3) Execute EXEC, SIREAD and SIEVAL phases.

(4) If the result of SIEVAL is true, lock at Q sites, execute UPDATE phase and unlock by 2PC, else do 5.

(5) Reject transaction, unlock at $(P + R)$ sites and terminate.

The number of messages generated by the *CR1DS* algorithm at each step is:

(1) $2nC(R + S)$
(2) $3n[1](P + Q + C^*R)$
(3) $2n[2](1 - C)(R + S)$ (messages due to the transfer of data for SI validation to one site)
(4) $3m[2](P + Q + R)$
(5) $2(n[2] - m[2])(P + R)$.

The cost of *CR1DS* algorithm is:

$$C[cr1ds] = 2nC(R + S) + 3n[1](P + Q + C^*R) + 2n[2](1 - C)(R + S) + 3m[2](P + Q + R) + 2(n[2])(P + R). \tag{18}$$

Algorithm CR2DS *for* CR2 *pre-execution-run time SI enforcement under* d–2*PL* & 2*PC*.

(1) Lock at $C(R + S)$ sites and execute SI test.

(2) If the result of SI test is false then reject transaction, unlock $C(R + S)$ sites and terminate, else do 3.

(3) Execute EXEC, SIREAD and SIEVAL phases.

(4) If the result of SIEVAL is false then reject transaction, unlock $(P + R)$ sites and terminate, else do 5.

(5) Lock at Q sites, execute transaction UPDATE phase and unlock by 2PC, and terminate.

The number of messages generated by the CR2DS algorithm at each step is:

(1) $2nC(R+S)$
(2) $2n[1]C(R+S)$
(3) $2n[2](1-C)(R+S)$
(4) $2(n[2]-m[2])(P+R)$
(5) $3m[2](P+Q+R)$.

The cost of CR2DS algorithm is:

$$\begin{aligned}C[cr2ds] &= 2nC(R+S)+2n[1]C(R+S)\\ &\quad +2n[2](1-C)(R+S)\\ &\quad +2(n[2]-m[2])(P+R)\\ &\quad +3m[2](P+Q+R). \qquad (19)\end{aligned}$$

3.8.1. *Cost comparison.* Now that the costs of SI enforcement methods in distributed DBS based on slow network and with d-2PL & 2PC concurrency control has been derived it is useful to compare them.

Assertion 7. The cost of CR1 pre-execution-run time SI enforcement is always lower than the cost of run time SI enforcement, i.e.

$$C[cr1ds] \leqslant C[rtds].$$

Proof

To prove that

$$C[cr1ds] \leqslant C[rtds]$$

we substitute from (16) and (19) and obtain

$$3Rm[1]+2n[1](R+S) \geqslant 0 \qquad (21)$$

which is always true.

Assertion 8. The cost of CR2 pre-execution-run time SI enforcement is always lower than the cost of CR1 pre-execution-run time SI enforcement, i.e.

$$C[cr2ds] \leqslant C[cr1ds]$$

Proof

To prove that

$$C[cr2ds] \leqslant C[cr1ds]$$

we substitute from (18) and (19) and obtain

$$C \leqslant \frac{2P-R}{2S-R}. \qquad (22)$$

Since $S \leqslant P$ and $0 < C \leqslant 1$ then (22) is true.

Assertion 9. The cost of pre-execution time SI enforcement is always lower than the cost of CR2 pre-execution-run time SI enforcement, i.e.

$$C[ctds] \leqslant C[cr2ds].$$

Proof

To prove that

$$C[ctds] \leqslant C[cr2ds]$$

we substitute from (17) and (19) and obtain

$$\frac{C(2n[2]-m[2])-n[2]}{n[2]-m[2]} \leqslant \frac{P+R}{R+S} \leqslant 1 \text{ (as } S \leqslant P) \qquad (23)$$

(23) *reduces to* $C \leqslant 1$, which is true.

In order to establish the cost differences among SI enforcement methods in distributed DBS based on slow network when $k < 1$ we use three cases of parameter setting. In scenario 1 we derive the SI enforcement costs assuming low intensity of SI enforcement ($S = R = 0.1$), in scenario 2 we assume medium intensity of SI enforcement ($S = R = 1$), in scenario 3 we assume high intensity of SI enforcement ($S = R = 2.5$). In all three cases we assume $P = Q = 3$, $n = 100$, $m = 95$, $n[2] = 97$, $n[1] = 3$. For each scenario we derive SI enforcement cost as a function of SI test efficiency C. The results for scenario 1–3 are shown in Figs. 12–14.

Next three figures show SI enforcement cost as a function of SI enforcement intensity (i.e. $S =$

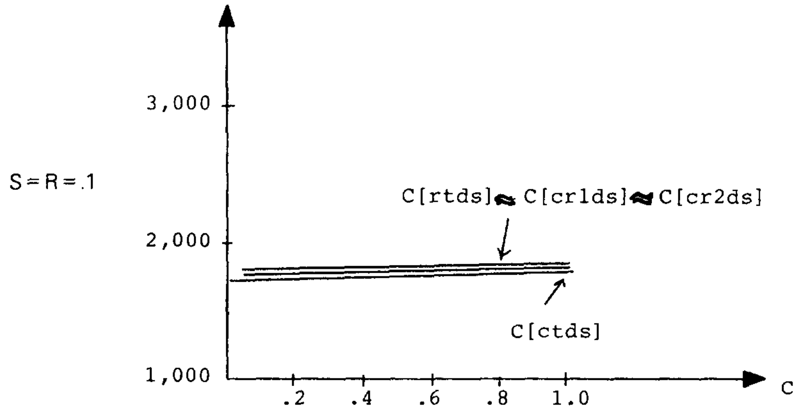

Fig. 12.

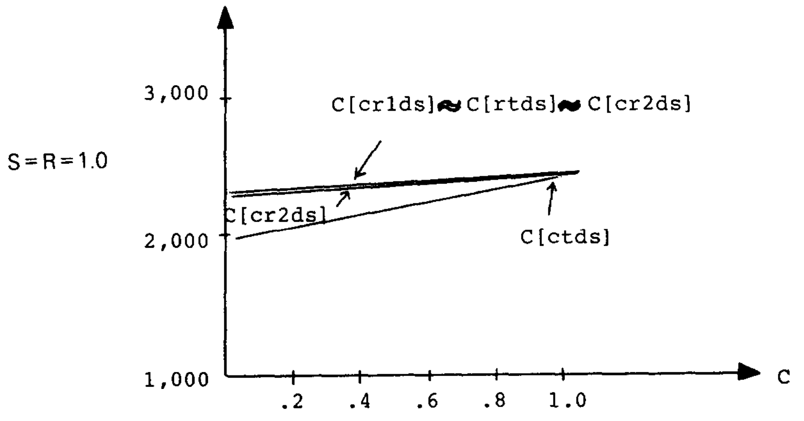

Fig. 13.

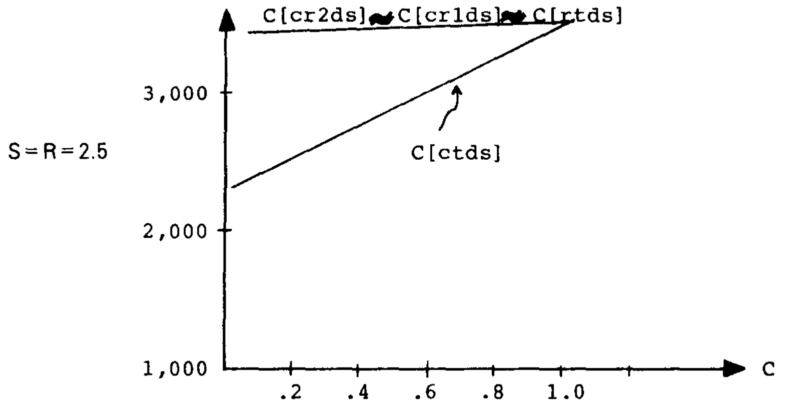

Fig. 14.

$R \in \langle 0.1, 3 \rangle)$ for three levels of SI test efficiency C; $C = 0.1$ (Fig. 15), $C = 0.5$ (Fig. 16), $C = 1$ (Fig. 17).

To conclude the SI enforcement cost analysis in distributed DBS based on slow network we observe that two SI enforcement methods, run time and CR1 pre-execution-run time, have almost identical costs. CR2 pre-execution-run time SI enforcement has visible but not significantly lower cost. Pre-execution time SI enforcement exhibits the lowest cost and thus it can represent considerable cost savings and increased DBS performance. We observe here that the derived conclusions apply to any type of distributed DBS based on slow network, i.e. to fully replicated, partially replicated or nonredundant databases. This is because the obtained results do not depend on the number of sites at which transaction writes only, i.e. on Q.

3.9 *Cost of SI enforcement in distributed DBS based on a fast or local network*

In this section we derive the cost of different SI enforcement methods for distributed DBS based on fast or local networks. This means that we consider both the cost of secondary memory accesses and the communication cost as equally significant. Thus we use cost formulas derived for distributed DBS based on the slow network and we expand them to include the cost of secondary memory accesses. However, we consider only distributed two-phase locking (2PL) with two phase commit (2PC) concurrency control mechanism.

In our analysis we assume that the speed of secondary memory channels is comparable to that of the interconnecting network. Of course, should the speed of the network become very small with respect to secondary memory channels, as might be the case in some ultra-fast local networks, then such distributed DBS becomes in our analysis equivalent to the centralized DBS.

The cost of run time SI enforcement is derived from the *RTDS* algorithm. We list here only the number of messages generated at each step and the number of database accesses at each step of the *RTDS* algorithm.

(1) $n(X + W)$
(2) $2n(R + S)$
(3) $2(n - m)(P + R)$
(4) $3m(P + Q + R) + mY$.

The cost of modified *RTDS* algorithm, now called *RTDF* algorithm, is:

$$C[rtdf] = 2n(R+S) + 2(n-m)(P+R) + 3m(P+Q+R) + n(X+W) + mY. \quad (25)$$

The cost of pre-execution time SI enforcement can be derived in a similar manner from the *CTDS* algorithm.

(1) $2nC(R+S) + kn(Z+W)$
(2) $2C(n-m)(R+S)$
(3) $m(X-k*Z)$
(4) $3m(P+Q+R)+mY$.

The cost of modified *CTDS* algorithm, now called *CTDF* algorithm, is:

$$C[ctdf] = 2nC(R+S) + 2C(n-m)(R+S) + 3m(P+Q+R) + kn(Z+W) + M(X-kZ+Y). \quad (26)$$

The cost of CR1 pre-execution-run time SI enforcement is derived from CR1DS algorithm and the messages it generates at each step as follows:

(1) $2nC(R*S) + nk(Z+W)$
(2) $3n[1](P+Q+R*C) + n[1](X-k*Z+Y)$
(3) $2n[2](1-C)(R+S) + n[2](X-Z*k) + (1-k)n[2]W$
(4) $3m[2](P+Q+R) + m[2]Y$
(5) $2(n[2]-m[2])(P+R)$.

The cost of modified CR1DS algorithm, now called CR1DF algorithm, is:

$$C[cr1df] = 2nC(R+S) + 3n[1](P+Q+R*C) + 2n[2](1-C)(R+S) + 3m[2](P+Q+R) + 2(n[2]-m[2])(P+R) + nk(Z+W) + n[1](X-Z*k+Y) + n[2](X-Z*k) + (1-k)n[2]W + m[2]Y. \quad (27)$$

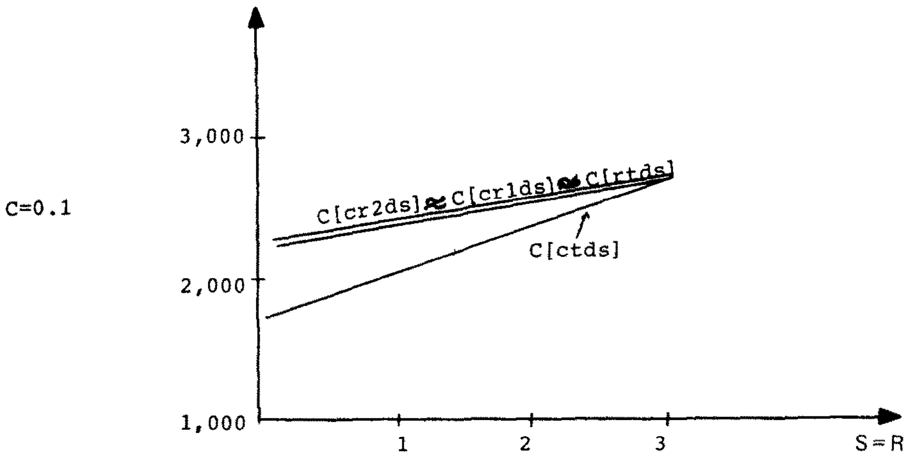

Fig. 15.

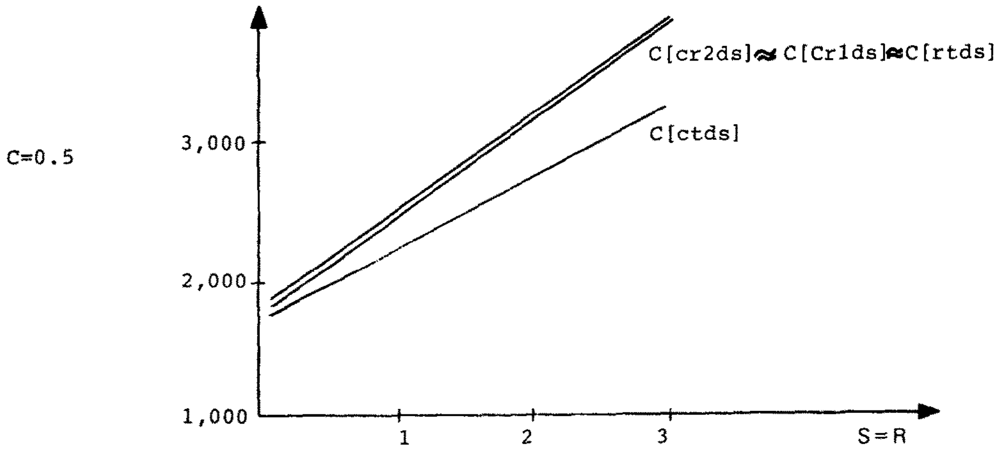

Fig. 16.

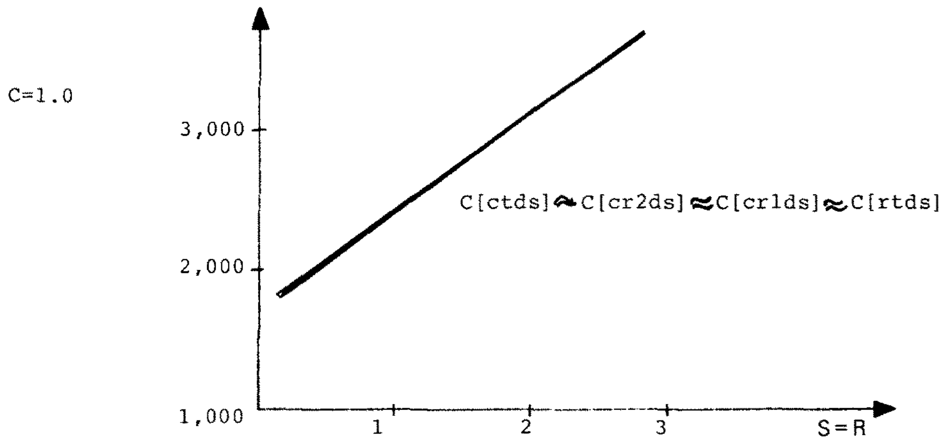

Fig. 17.

The cost of CR2 pre-execution-run time SI enforcement is derived from the CR2DS algorithm by adding secondary memory accesses to the number of messages generated at each step as follows.

(1) $2nC(R+S)+nk(Z+W)$
(2) $2n[1]C(R+S)$
(3) $2n[2](1-C)(R+S)+n[2](X-k*Z)$
$\quad +(1-k)n[2]W$
(4) $2(n[2]-m[2])(P+R)$
(5) $3m[2](P+Q+R)+m[2]Y.$

The cost of modified CR2DS algorithm, now called CR2DF, is:

$$\begin{aligned} C[cr2df] = {} & 2nC(R+S)+2n[1]C(R+S) \\ & +2n[2](1-C)(R+S) \\ & +2(n[2]-m[2])(P+R) \\ & +3m[2](P+Q+R)+nk(Z+W) \\ & +n[2](X-k*Z)+(1-k)n[2]W \\ & +m[2]Y. \end{aligned} \tag{28}$$

3.9.1 *Cost comparison.* We have already shown that in the centralized DBS and in the distributed DBS based on slow network

$$C[rt] \geqslant C[cr1] \geqslant C[cr2] \geqslant C[ct]. \tag{29}$$

Since the cost of SI validation in the distributed DBS based on fast network is obtained by adding the costs of each SI enforcement method in centralized DBS and distributed DBS based on slow network then the relation (29) holds for distributed DBS based on fast network as well.

In order to show the cost differences among SI enforcement methods we compute the cost of each for three cases. In scenario 1 we assume low intensity of SI enforcement by setting $S=R=Z=W=0.1$. In scenario 2 we assume medium intensity of SI enforcement by setting $Z=W=2$ and $S=R=1$. In scenario 3 we assume high intensity of SI enforcement by setting $S=R=2.5$ and $Z=W=4$. Other parameter settings are $X=5$, $Y=3$, $P=Q=3$, $n=100$, $m=95$, $n[2]=97$, $n[1]=3$. For each scenario we compute cost as a function of SI test efficiencies k and C. The results for scenario 1–3 are shown in Figs. 18–20.

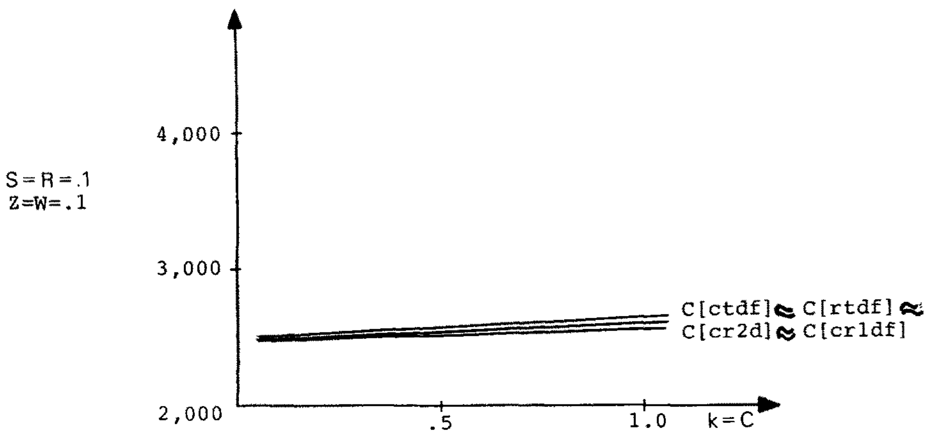

Fig. 18.

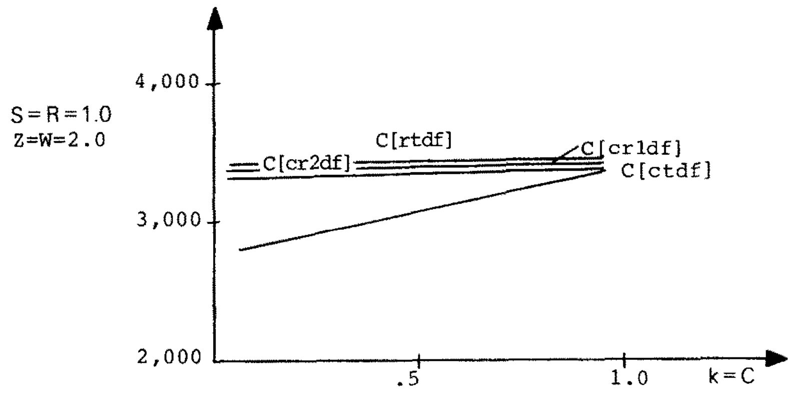

Fig. 19.

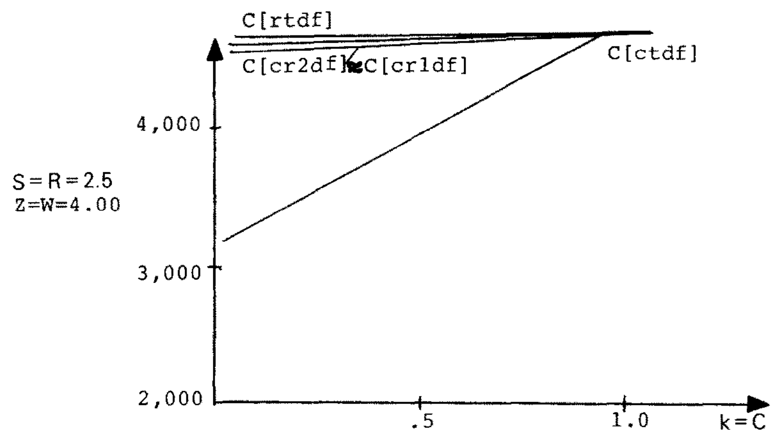

Fig. 20.

To conclude the SI enforcement cost analysis in distributed DBS based on fast network we observe that the run time, CR1 and CR2 pre-execution-run time SI enforcement methods have essentially the same cost. However, the pre-execution time SI enforcement can have considerably lower SI overhead. It is worth to mention that if SI tests are not efficient, i.e. $k = C = 1$, then all SI enforcement methods have the same cost. This result might be of considerable importance for the design of concurrency control and recovery mechanisms for centralized but in particular for distributed DBMS's.

4. CONCLUSIONS

In the first part of this paper we have investigated several heuristic techniques for SI enforcement and we have described in which way they can decrease SI enforcement cost. In the second part we have quantitatively analyzed several ways of sequencing and interleaving transaction execution and transaction validation processes. Such analysis consists of developing SI enforcement algorithms and counting a number of messages and/or database data accesses generated by each. We have analyzed several SI enforcement algorithms for centralized and distributed DBS based on slow and fast networks and have shown when they can result in different costs. In particular we have shown that the pre-execution time SI enforcement can have significantly lower cost then other SI enforcement methods whose costs are quite similar.

Acknowledgement—The author wishes to express his appreciation to the referees for their comments.

REFERENCES

[1] D. Z. Badal and G. J. Popek: Cost and performance analysis of semantic integrity validation methods. *Proc. ACM SIGMOD 79 Int. Conf. on Management of Data*, Boston, pp. 109–115 (May 1979).

[2] D. Z. Badal: On efficient monitoring of database assertions in distributed databases. *Proc. 4th Berkeley Conf. on Distributed Data Management and Computer Networks*, San Francisco, pp. 125–137 (Aug. 1979).

[3] O. P. Buneman and E. K. Clemons: Efficiently monitoring relational databases. *ACM Trans. on Database Systems* 4, 368–383 (1979).

[4] K. P. Eswaran and D. D. Chamberlin: Functional specification of a trigger subsystem in an integrated database system. IBM Res. Rep. RJ 1601 (June 1975).

[5] K. P. Eswaran: Specifications, implementations and interactions of a trigger subsystem in an integrated database system. IBM Res. Rep. RJ 1820 (Nov. 1976).

[6] K. P. Eswaran *et al*: The notions of consistency and predicate locks in database system. *CACM* **19**, 624–633 (1976).

[7] J. J. Florentin: Consistency auditing of databases. *Comput. J.* **17**, 52–58 (1974).

[8] J. Gray *et al.*: Granularity of locks and degrees of consistency in a shared database. *Modelling in Data Base Management Systems* (Edited by G. M. Nijssen), pp. 365–395. North Holland, Amsterdam (1976).

[9] J. Gray: Notes on data base operating systems. IBM Res. Rep. RJ 2188 (Feb. 1978).

[10] M. M. Hammer and S. K. Sarin: Efficient monitoring of database assertions. *Proc. ACM SIGMOD* 78 *Int. Conf. on Management of Data*, Dallas, pp. 38–48 (June 1978).

[11] B. W. Lampson and M. E. Sturgis, Crash recovery in a distributed storage system. XEROX PARC Res. Rep. (July 1976).

[12] B. G. Lindsay *et al.*: Notes on distributed databases. IBM Res. Rep. RJ2571 (July 1979).

[13] R. A. Lorie: Physical integrity in a large segmented database. *ACM TODS* **2**, 91–104 (1977).

[14] C. Machgeles: A procedural language for expressing integrity constrains in the coexistence model. *Modelling in Data Base Management Systems* (Edited by G. M. Nijssen), pp. 293–301. North-Holland, Amsterdam (1976).

[15] D. J. McLeod: High level expression of semantic integrity specifications in a relational data base system. MIT/LCS/TR-165 (Sept. 1965).

[16] N. Minsky: On interaction with data bases. *Proc. ACM SIGFIDET Workshop on Data Description, Access, and Control.* ACM, New York (1974).

[17] D. G. Severance and G. M. Lohman: Differential files: Their application to the maintenance of large databases. *ACM TODS* **1**, 256–267 (1976).

[18] M. Stonebraker: High level integrity assurance in relational data management systems. Electr. Res. Lab. Memo ERL-M473, UC Berkeley (Aug. 1974).

[19] M. Stonebraker: Implementation of integrity constraints and views by query modification. Electronics Res. Lab. Memo ERL-M514, UC Berkeley (March 1975).

[20] M. Stonebraker and E. Neuhold: A distributed data base version of INGRES. Electronics Res. Lab. Memo ERL-M612, UC Berkeley (Sept. 1976).

[21] H. Weber: A semantic model of integrity constraints on a relational data base. *Modelling in Data Base Management Systems* (Edited by G. M. Nijssen), pp. 269–293. North-Holland, Amsterdam (1976).

[22] M. M. Zloof: Query by example. IBM Res. Rep. RC 4917 (July 1974).