ISSN: 0128-7680
Pertanika J. Sci. & Technol. **10(1)**: 35-54 (2002)

Cost and Performance Analysis of Integrity Validation Techniques for a Distributed Database

Hamidah Ibrahim
Department of Computer Science
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
43400 UPM, Serdang, Selangor, Malaysia
e-mail: hamidah@fsktm.upm.edu.my

Received: 14 December 2000

ABSTRAK

Satu masalah utama dengan penggunaan kekangan integriti untuk mengawasi integriti pangkalan data yang berubah secara dinamik ialah kos penilaiannya. Kos ini yang berkait dengan prestasi mekanisme pemeriksaan ialah ukuran kuantitatif yang utama yang harus diselia dengan teliti. Kami telah membangunkan satu subsistem kekangan integriti untuk suatu pangkalan data teragih hubungan yang mengandungi beberapa teknik yang diperlukan untuk pemeriksaan kekangan secara efisien, terutamanya di dalam persekitaran teragih di mana pengagihan data adalah bebas kepada domain aplikasi. Di dalam makalah ini, kami akan menunjukkan bagaimana teknik-teknik ini telah dengan efektifnya mengurangkan kos pemeriksaaan kekangan di dalam persekitaran teragih.

ABSTRACT

A principal problem with the use of integrity constraints for monitoring the integrity of a dynamically changing database is their cost of evaluation. This cost which is associated with the performance of the checking mechanisms is the main quantitative measure which has to be supervised carefully. We have developed an integrity constraint subsystem for a relational distributed database (SICSDD) which consists of several techniques that are necessary for efficient constraint checking, particularly in a distributed environment where data distribution is transparent to the application domain. In this paper, we will show how these techniques have effectively reduced the cost of constraint checking in such a distributed environment.

Keywords: Distributed database, integrity constraints, constraint checking

INTRODUCTION

A database state is said to be consistent if the database satisfies a set of constraints, called *semantic integrity constraints*. Integrity constraints specify those configurations of the data that are considered semantically correct. Any update operation (insert, delete or modify) or transaction (sequence of updates) that occurs must not violate these constraints. Thus, a fundamental issue concerning integrity constraints is constraint checking, that is the process of ensuring that the integrity constraints are satisfied after the database has been updated.

Much attention has been paid to the maintenance of integrity in centralized databases over the last decade. A naive approach to constraint checking is to perform the update and then check whether all the integrity constraints are satisfied in the new database state. This method, termed *brute force checking*, is very expensive, impractical and can lead to prohibitive processing costs (Hsu and Imielinskin 1985; Mazumdar 1993; Qian 1989). Enforcement is costly because the evaluation of all integrity constraints requires accessing large amounts of data which are not involved in the database update state transition. Researchers have suggested that constraint checking can be optimized by exploiting the fact that the constraints are known to be satisfied prior to an update. This strategy known as *incremental integrity checking*, avoids redundantly checking constraints that are satisfied in the database before and are not affected by the update operation. It is the basis of most current approaches to integrity checking in databases. Another strategy is to simplify the constraint formulae so that *less data are accessed* in order to determine the truth of a constraint. Under the assumption that the set of initial constraints, *IC*, is known to be satisfied in the state before an update, simplified forms of *IC*, say *IC'*, are constructed such that *IC* is satisfied in the new state if and only if *IC'* is satisfied, and the evaluation cost of *IC'* is less than or equal to the evaluation cost of *IC*. This strategy is referred to as *constraint simplification* and the simplified forms of these constraints are referred to as integrity tests. This approach conforms with the admonition of Nicholas (1982) to concentrate on the problem of finding *good*[1] constraints. Various simplification techniques have been proposed where integrity tests are derived from the syntactic structure of the constraints and the update operations (Gupta and Widom 1993; Hsu and Imielinskin 1985; McCune and Henschen 1989; Nicholas 1982; Simon and Valduriez 1986). Researchers in this area have focussed solely on the derivation of efficient integrity tests, claiming that they are cheaper to enforce and reduce the amount of data accessed, thus reducing the cost of integrity constraint checking. Three different types of integrity test are defined in (McCune and Henschen 1989), namely: *sufficient tests, necessary tests* and *complete tests.*

Although this research effort has yielded fruitful results that have given centralized systems a substantial level of reliability and robustness with respect to the integrity of their data, there has so far been little research carried out on integrity issues for distributed databases. Devising an efficient algorithm for enforcing database integrity against updates is more crucial in a distributed environment. The reasons for this are described in (Barbara and Garcia-Molina 1992; Mazumdar 1993; Qian 1989; Simon and Valduriez 1986). The brute force strategy of checking constraints is worse in the distributed context since the checking would typically require data transfer as well as computation leading to complex algorithms to determine the most efficient approach. Allowing an update to execute with the intention of aborting it at commit time

[1] Good is intended to mean easy to maintain and easy to check (Nicholas 1982)

in the event of constraint violation is also inefficient since rollback and recovery must occur at all sites in which the update participated.

A principal problem with the use of integrity constraints for monitoring the integrity of a dynamically changing database is their cost of evaluation. This cost which is associated with the performance of the checking mechanisms is the main quantitative measure which has to be supervised carefully. Different criteria have been used to assess this performance such as the time to check the validity of the constraints against updates. Generally, an efficient constraint checking strategy tries to minimize the utilization of the computing resources involved during the checking activities. A common goal addressed by previous researchers in this field is to propose constraint simplification strategies which manage to derive a better set of constraints than the initial set. A simplification strategy is said to be efficient if the evaluation of the generated simplified forms has effectively reduced the cost of integrity checking compared to the evaluation of the initial constraints. Based on the following studies (Bernstein and Blaustein 1981; Gupta and Widom 1993; Hsu and Imielinskin 1985; Mazumdar 1993; Nicholas 1982; Qian 1988; Simon and Valduriez 1986), the cost of evaluating an integrity constraint includes the following *main* components: (i) the amount of data accessed (locally or non-locally for a distributed database) – this is related to the checking space of the integrity constraint (Hsu and Imielinskin 1985; Qian 1988); (ii) the amount of data transferred across the network; and (iii) the number of sites involved, which is interrelated with (ii) above. For a centralized database, the main emphasis is on minimizing the amount of data accessed or the checking space. In distributed databases, where many sites are involved, the amount of data transferred across the network and the number of sites involved must be minimized too. Most of the authors cited consider a single cost component due to the difficulty in assigning suitable weights to all cost components.

Bernstein and Blaustein (1981) and Nicholas (1982) used the following two intuitive arguments as a justification that the evaluation of a simplified constraint produced by their algorithm has effectively reduced the cost of integrity checking compared to the evaluation of the corresponding initial constraint: (i) the more constants that are substituted for variables in a given constraint, the more selective is the constraint, and so the easier it should be to evaluate and therefore the cheaper the evaluation; and (ii) the simplified forms are derived on a minimal substate of the database state and so involve less data access. Hsu and Imielinskin (1985) measured the simplicity of an integrity constraint by the notion of its checking space. The *checking space* of a constraint $IC(v_1, v_2, \ldots, v_n)$ is defined as $v_1 \times v_2 \times \ldots \times v_n$ where x is the cartesian product operator, and the v_i's are the range variables in the *IC*. A constraint *IC-i* is said to be *simpler* than a constraint *IC-j* if the checking space of *IC-i* is smaller than the checking space of *IC-j*. The checking space of a constraint is a rough measure of the complexity of its evaluation and the number of variables it has to access. This measurement is later used by other authors (Qian 1988). Simon and Valduriez (1986) claimed that their simplification method minimized the cost of integrity

checking since only data subject to update are evaluated. Thus they are reducing the checking space by removing data that is known to be unaffected by the change. Mazumdar (1993) proposed a metric called scatter, σ, to capture the amount of non-local access necessary to evaluate a constraint in a distributed database.

We have developed an integrity constraint subsystem for a relational distributed database (SICSDD). The subsystem provides complete functionality and an efficient strategy for constraint enforcement. Complete functionality is attained through a modular and extensible architecture in which several techniques are incorporated. These techniques are necessary to achieve efficient constraint enforcement, particularly in a distributed database. By database distribution we mean that a collection of data which belongs logically to the same system is physically spread over the sites (nodes) of a computer network where intersite data communication is a critical factor affecting the system's performance. In this paper, we will show how the SICSDD techniques have effectively reduced the cost of constraint checking in a distributed environment. We do this by analysing and comparing the generated simplified forms to their respective initial constraints with respect to the amount of data that has to be accessed, the amount of data transferred across the network and the number of sites that may be involved during the evaluation of these constraints/ simplified forms. In general, our strategy reduces the amount of data needing to be accessed since only fragments of relations subject to update are evaluated. The amount of data transferred across the network and the number of sites that may be involved are minimized by evaluating the simplified forms at the target site, i.e. the site where an update is to be performed.

PRELIMINARIES

Our approach has been developed in the context of relational databases (Date 1995), which can be regarded as consisting of two distinct parts, namely: an intensional part and an extensional part. A database is described by a database schema *D*, which consists of a finite set of relation schemas, $<R_1, R_2, \ldots, R_m>$. A relation schema is denoted by $R(A_1, A_2, \ldots, A_n)$ where R is the name of the relation (predicate) with n-arity and the A_i'*s* are the attributes of *R*. Let $\mathrm{dom}(A_i)$ be the domain values for attribute A_i. Then, an instance of *R* is a relation $\Re$ which is a finite subset of cartesian product $\mathrm{dom}(A_1) \times \ldots \times \mathrm{dom}(A_n)$. A database instance is a collection of instances for its relation schemas. A relational distributed database schema is described as a quadruple (*D, IC, FR, AS*) where *IC* is a finite set of integrity constraints, *FR* is a finite set of partitioning rules and *AS* is a finite set of allocation schemas.

The database integrity constraints considered in this paper are *state constraints* which include: domain (*IC-1*), key (*IC-2, IC-3*), referential (*IC-4*) and general semantic integrity constraints (*IC-5, IC-6*). They are expressed in prenex conjunctive normal form with the range restricted property (McCune and Henschen 1989; Nicholas 1982). A conjunct (literal) is an atomic formula of

the form $R(u_1, u_2, \ldots, u_k)$ where R is a k-ary relation name and each u_i is either a variable or a constant. A positive atomic formula (positive literal) is denoted by $R(u_1, u_2, \ldots, u_k)$ whilst a negative atomic formula (negative literal) is prefix by $\neg$. An (in)equality is a formula of the form u_1 *OP* u_2 (prefix with $\neg$ for inequality) where both u_1 and u_2 can be constants or variables and $OP \in \{<, \leq, >, \geq, < >, =\}$.

A set of fragmentation rules, *FR*, specifies the set of restrictions, C_i, that must be satisfied by each fragment relation $\Re_i$. These rules introduce a new set of integrity constraints and therefore have the same notation as *IC*. We assume that the fragmentation of relations satisfies the completeness, the disjointness and the reconstructability properties (Ozsu and Valduriez 1991). An allocation schema locates a fragment relation, $\Re_i$, to one or more sites. Throughout this paper the same example *emp_dept* database is used, as given in *Fig. 1*. Here we assume that the relation *emp* is first vertically fragmented into emp_1 on attribute set A_1 = {*eno, ename, eaddress*} and emp_2 on attribute set A_2 = {*eno, dno, ejob, esal*};

Schema: emp(eno,ename,eaddress,dno,ejob,esal)
dept(dno,dname,mgrno,mgrsal)
Integrity Constraints (Global Constraints):
'A specification of valid salary'
IC-1 : $(\forall u \forall v \forall w \forall x \forall y \forall z)(emp(u, v, w, x, y, z) \rightarrow (z > 0))$
'Every employee has a unique eno'
IC-2 : $(\forall u \forall v1 \forall v2 \forall w1 \forall w2 \forall x1 \forall x2 \forall y1 \forall y2 \forall z1 \forall z2)$ $(emp(u, v1, w1, x1, y1, z1) \wedge emp(u, v2, w2, x2, y2, z2) \rightarrow (v1 = v2) \wedge (w1 = w2) \wedge (x1 = x2) \wedge (y1 = y2) \wedge (z1 = z2))$
'Every department has a unique dno'
IC-3 : $(\forall w \forall x1 \forall x2 \forall y1 \forall y2 \forall z1 \forall z2)(dept(w, x1, y1, z1) \wedge dept(w, x2, y2, z2) \rightarrow (x1 = x2) \wedge (y1 = y2) \wedge (z1 = z2))$
'The dno of every tuple in the *emp* relation exists in the *dept* relation'
IC-4 : $(\forall r \forall s \forall t \forall u \forall v \forall w \exists x \exists y \exists z)(emp(r, s, t, u, v, w) \rightarrow dept(u, x, y, z))$
'Every manager in department *D1* earns more than 4000 pounds'
IC-5 : $(\forall w \forall x \forall y \forall z)(dept(w, x, y, z) \wedge (w = D1) \rightarrow (z > 4000))$
'Every employee must earn ≤ to the manager in the same department'
IC-6 : $(\forall r \forall s \forall t \forall u \forall v \forall w \forall x \forall y \forall z)(emp(r, s, t, u, v, w) \wedge dept(u, x, y, z) \rightarrow (w \leq z))$
Fragmentation Rules:
FR-1: $(\forall w \forall x \forall y \forall z)(emp_{21}(w, x, y, z) \rightarrow (x = D1))$
FR-2: $(\forall w \forall x \forall y \forall z)(emp_{22}(w, x, y, z) \rightarrow (x = D2))$
FR-3: $(\forall w \forall x \forall y \forall z)(dept_1(w, x, y, z) \rightarrow (w = D1))$
FR-4: $(\forall w \forall x \forall y \forall z)(dept_2(w, x, y, z) \rightarrow (w = D2))$

Fig. 1: The emp_dept intensional database

emp_2 and *dept* are horizontally fragmented into two fragments emp_{2i} and $dept_i$, respectively, with predicates *dno* = *D*1 and *dno* = *D*2.

The key problem in integrity checking is how to efficiently evaluate constraints. An accurate way of measuring the evaluation of the generated fragment constraints/simplified forms is a high priority so that they can be compared fairly and realistically. In this paper, we will evaluate the derived set of fragment constraints/simplified forms with respect to the following components. We use the symbol $C(R_1, R_2, \ldots, R_n)$ to denote the set of relations or fragment relations specified in the constraint/simplified form *C*.

A – provides an estimate of the amount of data accessed, which is related to the number and the size of the relations or fragment relations specified in a

given constraint/simplified form. This measurement indirectly indicates the size of the checking space. It is based on the following formula: $A_{C(\Re 1, \Re 1, ..., \Re n)} = \delta\Re 1 + \delta\Re 2 + ... + \delta\Re n$ where the $\delta\Re i$'s are the relations or fragment relations specified in the C and $\delta\Re i$ is the size of $\Re i$. For a compound constraint $C = \Lambda^{m}_{i=1} C_i$, where C_i is a simple constraint, the amount of data accessed is measured as follows: $\Sigma^{m}_{i=1} A_{ci}$. For a disjunction of constraints, $C = V^{m}_{i=1} C_i$, the amount of data accessed is measured as follows: $[A_{min}, \Sigma^{m}_{i=1} A_{ci}]$ which is a range where A_{min} ($\Sigma^{m}_{i=1} A_{ci}$, respectively) is the minimum (maximum, respectively) amount of data that might need to be accessed.

τ – provides an estimate of the amount of data transferred across the network. It is measured based on the following formula, $\tau = \Sigma^{n}_{i=1} dt_i$, where dt_i is the amount of data transferred from site i to the target site.

σ - gives a rough measurement of the amount of non-local access necessary to evaluate a constraint/simplified form, and is taken from Mazumdar (1993). This is measured by analysing the number of sites that might be involved in validating the constraint. In general, the formula applied is as follows: $\sigma_C = |site(C)|$ where σ_C is the locality of C, i.e. the number of sites involved, and $site(C)$ is obtained by the following formula.

FOR each distinct relation or fragment relation, fr_i for $i = \{1, 2, ..., k\}$ in C DO
site$(C) = \{j : j \in$ sites Λ fr_i is allocated at site j where sites $\in \{1, 2, ..., n\}\}$

For a compound constraint $C = \Lambda^{m}_{i=1} C_i$, where C_i is a simple constraint, the locality of these constraints is simply obtained from the maximum number of sites that are involved in evaluating one of the constraints, i.e. $\sigma_C = max(\sigma_{C1}, \sigma_{C2}, ..., \sigma_{Cm})$. The locality of an integrity test, given an update operation, gives a rough measurement of the amount of non-local access necessary for verifying if the integrity test is being satisfied or violated by the update operation. This is measured by analysing the number of sites that might be involved in validating the test. In general, the formula applied is as follows:

$$\sigma_{Ti} = \begin{cases} |site(T_i)| & if\ site(U_i) \subseteq site(T_i) \quad (1) \\ |site(T_i)| + 1 & otherwise \end{cases} \qquad (1)$$

where σ_{Ti} is the locality of an integrity test, T_i, with respect to a given update operation, U_i, i.e. the number of sites involved. The site(T_i) and site(U_i) are obtained by the following formula.

FOR each distinct relation or fragment relation, fr_i for $i = \{1, 2, ..., k\}$ in T_i DO

$site(T_i) = \{j : j \in$ sites Λ fr_i is allocated at site j where sites $\in \{1, 2, \ldots, n\}\}$
$site(U_i) = \{l : l \in$ sites[2] Λ fr_{ui} is allocated at site l where sites $\in \{1, 2, \ldots, n\}$ and fr_{ui} is a fragment relation specified in $U_i\}$

Table 1 shows information relating to the *emp_dept* database. This information includes each relation $\Re$, its size $\delta\Re$, its fragment relation $\Re_i$ and their sizes $\delta\Re_i$. The fragment relations are derived based on the fragmentation schemas given earlier in this section. Case I and Case II in the table represent the sites where

TABLE 1
The *emp_dept's* relations and fragment relations

$\Re$	$\delta\Re$	$\Re_i$	$\delta\Re_i$	Case I[a]	Case II[a]
emp	E^b	emp_1	e_1	S_0	S_0
		emp_{21}	e_{21}	S_1	S_1
		emp_{22}	e_{22}	S_2	S_2
dept	D^c	$dept_1$	d_1	S_1	S_3
		$dept_2$	d_2	S_2	S_4

[a] $S_i \neq S_j$ where $i \neq j$
[b] $E = e_1 + \Sigma^2_{i=1} e_{2i} - \delta_{key}$ where δ_{key} is the size of the repeated primary key values
[c] $D = \Sigma^2_{i=1} d_i$

the fragment relations are allocated. Here Case I represents the reasonable case where the fragment relations constructed based on the same fragmentation rules are allocated to the same site. It is called the reasonable case as it should minimize the network traffic. Case II represents a worst case where each fragment relation is allocated to a different site of the network. It is a worst case in that network traffic is involved in any constraint evaluation that involves more than one fragment.

OUR OPTIMIZATION TECHNIQUES AND THEIR PERFORMANCES

The high execution cost of constraint enforcement is one of the major problems in the field of constraint handling (Grefen 1993). This cost can be substantially reduced not only by applying an efficient enforcement strategy but also by generating/evaluating a good set of integrity constraints. The techniques that are incorporated into our system seek to derive *efficient* sets of fragment constraints and a range of possible local tests [3]. Our techniques are identified

[2] As we assume that there is no replication of fragments across the network, for a given update operation, U_i there will be only one target site, i.e. the site where the update is to be performed.
[3] An integrity test is a local test if it can be evaluated at a single site, i.e. at the site where the update is to be performed, and a global test otherwise.

as *constraint preprocessing, constraint distribution* and *integrity test generation.* In this section, we will evaluate the derived set of fragment constraints/simplified forms with respect to the components described in the previous section.

Constraint Preprocessing Techniques

There are five steps that are applied which are performed by the following procedures (we use the notation $\vec{X}$ to indicate a finite set of variables, $x_1, x_2, ..., x_k$; and x_{key}, x_{join} and x_{ref} are the key, join and reference attribute, respectively). In general, the amount of data accessed for evaluating the set of fragment constraints, A_{FC-i}, derived by the procedures embodied in the constraint preprocessing techniques is less than the amount of data accessed for evaluating the initial constraint, A_{IC-i}, i.e. $A_{FC-i} < A_{IC-i}$. $\delta\Re_{min}$ is *minimum* ($\delta\Re_1, \delta\Re_2, ..., \delta\Re_n$).

constraint_transformation_procedure

This procedure transforms the constraints specification at the relational level (global constraints) into a constraints specification at the fragment level (fragment constraints). The objective of the transformation is to obtain a specification of constraints that can be straightforwardly used for constructing efficient enforcement algorithms. At this stage, the transformations are restricted to logically equivalent transformations, without considering any reformulation of the original constraints. Initially, each occurrence of a relation $\Re$ in a constraint is replaced by its n fragment relation, $\Re_i$. The transformation is one to many, which means that given a global constraint, the result of the transformation is a logically equivalent set of fragment constraints. There are four transformation rules which are applied during this process. These rules analyse the patterns in the prefixes of a constraint and the types of fragmentation that are being applied to the global relations. The proofs of these rules can be found in (Qian 1989) and are therefore omitted here.

Horizontal transformation rules:

i. $(\forall \vec{X})(R(\vec{X}) \rightarrow P(\vec{X})) \equiv \Lambda^n_{i=1}(\forall \vec{X})(R_i(\vec{X}) \; P(\vec{X}))$

ii. $(\exists \vec{X})(R(\vec{X}) \wedge P(\vec{X})) \equiv \mathrm{V}^n_{i=1}(\exists \vec{X})(R_i(\vec{X}) \wedge P(\vec{X}))$

Vertical transformation rules:

i. $(\forall \vec{X})(R(\vec{X}) \rightarrow P(\vec{X})) \equiv (\forall x_{key} \forall \vec{X}_1 \; ... \; \forall \vec{X}_n)(R_1(x_{key}, \vec{X}_1) \wedge ... \wedge R_n(x_{key}, \vec{X}_n) \rightarrow P(x_{key}, \vec{X}_1, ..., \vec{X}_n))$

ii. $(\exists \vec{X})(R(\vec{X}) \wedge P(\vec{X})) \equiv (\exists x_{key} \exists \vec{X}_1 ... \exists \vec{X}_n)(R_1(x_{key}, \vec{X}_1) \wedge ... \wedge R_n(x_{key}, \vec{X}_n) \wedge P(x_{key}, \vec{X}_1, ..., \vec{X}_n))$

To obtain a logically equivalent set of fragment constraints from a given global constraint which contains a relation $\Re$ and is fragmented by a mixed fragmentation, we repeatedly apply horizontal and vertical transformation rules in the same sequence of horizontal and vertical fragmentations to the global constraint to produce the fragment constraints. Example: The integrity constraint *IC-4* is transformed into the following set of fragment constraints.

$$FC\text{-}4_t : \Lambda_{i=1}^{2} V_{j=1}^{2} (\forall t \forall u \forall v \forall w \exists x \exists y \exists z)(emp_{2i}(t, u, v, w) \rightarrow dept_j(u, x, y, z))$$

As expected, evaluating the sets of fragment constraints derived by the constraint_transformation_procedure without further optimization is inefficient (with respect to the amount of data accessed) since the evaluation of these constraints generally will access all the fragment relations specified in these constraints. This is similar to the brute force strategy, but is only the initial stage of our application.

simplification_procedure

This procedure, which uses knowledge about the fragmentation of relations, reduces the number of fragment relations involved in the evaluation of a constraint. A set of fragment constraints derived from a global constraint can be simplified if the relations specified in the global constraint are fragmented on a join/reference attribute using the same fragmentation algorithm. Instead of deriving all combinations of fragment constraints, only compatible fragment constraints are constructed. Given global relations $\Re$ and S which are fragmented on a join/reference attribute into n and m fragment relations, respectively on the same fragmentation rules, then the following *simplification rules* are applied.

i. $\Lambda_{i=1}^{n} V_{j=1}^{m} (\forall x_{ref} \forall X^{-} \exists Y^{-})(R_i(x_{ref}, X^{-}) \rightarrow S_j(x_{ref}, Y^{-})) \equiv \Lambda_{i=1}^{n} (\forall x_{ref} \forall X^{-} \exists Y^{-})$
$(R_i(x_{ref}, X^{-}) \rightarrow S_i(x_{ref}, Y^{-}))$

ii. $\Lambda_{i=1}^{n} \Lambda_{j=1}^{m} (\forall x_{join} \forall X^{-} \forall Y^{-})(R_i(x_{join}, X^{-}) \Lambda S_j(x_{join}, Y^{-}) \ldots \rightarrow \ldots) \equiv$
$\Lambda_{i=1}^{n} (\forall x_{join} \forall X^{-} \forall Y^{-})(R_i(x_{join}, X^{-}) \Lambda S_i(x_{join}, Y^{-}) \ldots \rightarrow \ldots)$

iii. $V_{i=1}^{n} V_{j=1}^{m} (\exists x_{join} \exists X^{-} \exists Y^{-})(R_i(x_{join}, X^{-}) \Lambda S_j(x_{join}, Y^{-}) \ldots) \equiv$
$V_{i=1}^{n} (\exists x_{join} \exists X^{-} \exists Y^{-})(R_i(x_{join}, X^{-}) \Lambda S_i(x_{join}, Y^{-}) \ldots)$

Example: $FC\text{-}4_t$ above can be simplified since both relations *emp* and *dept* are fragmented on the reference attribute, i.e. *dno*, using the same fragmentation rules. The simplified set of fragment constraints are as shown below.

$$FC\text{-}4_{si} : \Lambda_{i=1}^{2} (\forall t \forall u \forall v \forall w \exists x \exists y \exists z)(emp_{2i}(t, u, v, w) \rightarrow dept_i(u, x, y, z))$$

The simplification_procedure simplifies a set of fragment constraints which reduces the number of joins/references between fragment relations required in the definition of the set of fragment constraints.

subsumption_procedure

This procedure attempts to obtain an improved set of fragment constraints by removing redundant fragment constraints from a set. A redundant fragment constraint is a constraint that can be implied (syntactically) from other existing fragment constraints. Thus, deleting a redundant fragment constraint from its set does not affect the unsatisfiability/satisfiability of the set since the truth of this constraint can be inferred from the truth of its identical or subsumed pair.

Even though a redundant set of constraints is semantically correct, excluding redundancy can improve the enforcement time. Example: Consider the following set of fragment constraints derived from *IC-3* by applying the transformation process constraint_transformation_procedure.

$$FC\text{-}3t:\ \Lambda_{i=1}^{2}\,\Lambda_{j=1}^{2}\ (\forall w \forall x1 \forall x2 \forall y1 \forall y2 \forall z1 \forall z2)\ (\mathrm{dept}_i(w, x1, y1, z1)\ \Lambda\ \mathrm{dept}_j(w, x2, y2, z2) \rightarrow (x1 = x2)\ \Lambda\ (y1 = y2)\ \Lambda\ (z1 = z2))$$

The sets $[(\forall w \forall x1 \forall x2 \forall y1 \forall y2 \forall z1 \forall z2)\,(dept_i(w, x1, y1, z1)\ \Lambda\ dept_j(w, x2, y2, z2) \rightarrow (x1 = x2)\ \Lambda\ (y1 = y2)\ \Lambda\ (z1 = z2))]$ and $[(\forall w \forall x1 \forall x2 \forall y1 \forall y2 \forall z1 \forall z2)\,(dept_j(w, x1, y1, z1)\ \Lambda\ dept_i(w, x2, y2, z2) \rightarrow (x1 = x2)\ \Lambda\ (y1 = y2)\ \Lambda\ (z1 = z2))]$ for $i \in \{1, 2\}$ and $j \in \{1, 2\}$ are mutually redundant. Removing the redundant fragment constraints will generate the following set of fragment constraints:

$$FC\text{-}3_{su}:\ \Lambda_{i=1}^{2}\,V_{j=i}^{2}\ (\forall w \forall x1 \forall x2 \forall y1 \forall y2 \forall z1 \forall z2)\ (dept_i(w, x1, y1, z1)\ \Lambda\ dept_j(w, x2, y2, z2) \rightarrow (x1 = x2)\ \Lambda\ (y1 = y2)\ \Lambda\ (z1 = z2))$$

The subsumption_procedure removes any redundant fragment constraints from a given set of fragment constraints, i.e. $\Lambda_{j=1}^{n} fc_j$ or $V_{j=1}^{m} fc_j$ or a set of fragment constraints constructed by both. Obviously, this set of fragment constraints is derived from the horizontal/mixed transformation rules.

contradiction_procedure

This procedure removes the fragment constraints produced by the constraint_transformation_procedure which contradict the fragmentation rules, i.e. it removes fragment constraints that are never satisfied. The contradiction_procedure designed by us is a specific-purpose theorem prover that resembles a resolution-based theorem prover similar to Henschen *et al.* (1984) and McCarrol (1995). It is mainly designed for investigating if a fragment constraint violates one of the existing fragmentation rules. Example: The fragment constraint $FC\text{-}3_{su}$ above, when $i \neq j$, contradicts the fragmentation rules *FR-3* and *FR-4*.

The contradiction_procedure eliminates fragment constraints from their sets if they contradict one of the existing fragmentation rules, i.e. $\Lambda_{j=1}^{n} fc_j$ or $V_{j=1}^{m} fc_j$ or a set of fragment constraints constructed by both. Obviously, this set of fragment constraints is derived from the horizontal/mixed transformation rules.

reformulation_procedure

The above procedures (the simplification_procedure, the subsumption_procedure and the contradiction_procedure) use knowledge about data fragmentation and analyse the syntax of the constraints to either remove inessential constraints (a redundant constraint or a constraint which contradicts the fragmentation rules) from the constraints set or reduce the scope of the

fragment relations specified in a constraint (the simplification-procedure). These procedures, which are based on syntactic criteria, do not check the possible occurrence of redundant semantics in the constraint set. The reformulation_procedure, which is based on semantic criteria, attempts to (i) remove redundant semantic constructs that may exist, and (ii) reformulate the set of fragment constraints derived so far into alternative forms which can be either an *antecedent* [4] or an *equivalent* form. In our approach, removing the redundant semantic constructs from a given fragment constraint is achieved by applying the substitution and absorption rules. This strategy makes the constraint easier to evaluate because more constants are substituted for the variables in the constraint which makes the constraint more selective. Example: Consider the following fragment constraint derived from *IC-5* by applying the transformation (constraint_transformation_procedure) and the contradiction (contradiction_procedure) processes.

$$FC\text{-}5_t : (\forall w \forall x \forall y \forall z)(dept_1(w, x, y, z) \wedge (w = \mathrm{D1}) \rightarrow (z > 4000))$$

The literal $w = D1$ is a redundant construct as this can be implied from *FR-3*. Removing this construct will generate the following fragment constraint.

$$FC\text{-}5_r : (\forall w \forall x \forall y \forall z)(dept_1(w, x, y, z) \rightarrow (\mathrm{z} > 4000))$$

The fragment constraint $FC\text{-}6_{t1}$ below which is derived from *IC-6* can be reformulated as *FC-6'* by using $FC\text{-}5_r$ derived above.

$$FC\text{-}6_{t1} : (\forall t \forall u \forall v \forall w \forall x \forall y \forall z)(emp_{21}(t, u, v, w) \wedge dept_1(u, x, y, z) \rightarrow (w \leq z))$$
$$FC\text{-}6' : (\forall t \forall u \forall v \forall w)(emp_{21}(t, u, v, w) \rightarrow (w \leq 4000))$$

Since we require that a fragment constraint which is derived by the reformulation process must be cheaper compared to the initial constraint, therefore the amount of data accessed for evaluating the derived constraint must be less than or equal to the amount of data accessed for evaluating the initial constraint.

Fig. 2 lists the sets of fragment constraints derived after applying the constraint preprocessing techniques to the initial constraints. Each *FC-i* is a semantically equivalent set to its initial constraint *IC-i* (except for *FC-6'* which is an antecedent of $FC\text{-}6_{t1}$).

Table 2 illustrates the amount of data needing to be accessed, A, by the sets of fragment constraints given in *Fig. 2* (except for *FC-6'*) which are derived by the procedures embodied in the constraint preprocessing techniques. In most cases the derived set of fragment constraints is better than the initial constraint which is similar to the brute force strategy, i.e. $A_{FC\text{-}i} < A_{IC\text{-}i}$, as shown by *FC-1, FC-*

[4] If an antecedent of a fragment constraint is satisfied this implies that the fragment constraint is satisfied but if the antecedent is falsified, then the fragment constraint has to be evaluated.

3, FC-4, FC-5 and *FC-6* in Table 2. $A_{FC\text{-}i} > A_{IC\text{-}i}$, only when $A_{IC\text{-}i}$ is specified over the attribute(s) of a global relation $\Re$ which is associated with all of its fragment relations which are constructed by a vertical fragmentation, as shown by *FC-2* in Table 2.

FC-1 : $\Lambda^2_{i=1}(\forall w \forall x \forall y \forall z)\ (emp_{2i}(w, x, y, z) \rightarrow (z > 0))$
FC-2 : $\Lambda^2_{i=1}(\forall u \forall v1 \forall v2 \forall w1 \forall w2 \forall x \forall y1 \forall y2 \forall z1 \forall z2)\ (emp_1(u, v1, w1) \Lambda emp_{2i}(u, x, y1, z1) \Lambda (emp_1(u, v2, w2) \Lambda emp_{2i}(u, x, y2, z2) \rightarrow (v1 = v2) \Lambda (w1 = w2) \Lambda (y1 = y2) \Lambda (z1 = z2))$ and $(\forall u \forall x1 \forall x2 \forall y1 \forall y2 \forall z1 \forall z2)$ $(emp_{2i}(u, x1, y1, z1) \rightarrow \neg(emp_{2i}(u, x2, y2, z2))$
FC-3 : $\Lambda^2_{i=1}(\forall w \forall x1 \forall x2 \forall y1 \forall y2 \forall z1 \forall z2)(dept_i(w, x1, y1, z1) \Lambda dept_i(w, x2, y2, z2) \rightarrow (x1 = x2) \Lambda (y1 = y2) \Lambda (z1 = z2))$
FC-4 : $\Lambda^2_{i=1}(\forall t \forall u \forall v \forall w \exists x \exists y \exists z)(emp_{2i}(t, u, v, w) \rightarrow dept_i(u, x, y, z))$
FC-5 : $(\forall w \forall x \forall y \forall z)(dept_1(w, x, y, z) \rightarrow (z > 4000))$
FC-6 : $\Lambda^2_{i=1}\ (\forall t \forall u \forall v \forall w \forall x \forall y \forall z)(emp_{2i}(t, u, v, w) \Lambda dept_i(u, x, y, z) \rightarrow (w \leq z))$
FC-6' : $(\forall t \forall u \forall v \forall w)\ (emp_{21}(t, u, v, w) \rightarrow (w \leq 4000))$

Fig. 2: The sets of fragment constraints derived by the constraint preprocessing techniques with respect to the global constraints and fragmentation rules given in Fig. 1

TABLE 2
Estimation of the amount of data accessed – the constraint preprocessing techniques

IC-i	$A_{IC\text{-}i}$	*FC-i*	$A_{FC\text{-}i}$	Comment
IC-1	E	*FC-1*	$\Sigma^2_{i=1} e_{2i}$	$A_{FC\text{-}1} < A_{IC\text{-}1}$
IC-2	$E + E$	*FC-2*	$\Sigma^2_{i=1}(2e_1 + 2e_{2i}) + e_{21} + e_{22}$	$A_{FC\text{-}2} > A_{IC\text{-}2}$
IC-3	$D + D$	*FC-3*	$\Sigma^2_{i=1} d_i + d_i$	$A_{FC\text{-}3} < A_{IC\text{-}3}$
IC-4	$E + D$	*FC-4*	$\Sigma^2_{i=1} e_{2i} + d_i$	$A_{FC\text{-}4} < A_{IC\text{-}4}$
IC-5	D	*FC-5*	d_1	$A_{FC\text{-}5} < A_{IC\text{-}5}$
IC-6	$E + D$	*FC-6*	$\Sigma^2_{i=1} e_{2i} + d_i$	$A_{FC\text{-}6} < A_{IC\text{-}6}$

The amount of data accessed by *FC-i* is determined by the assumption that all fragment constraints in *FC-i* are evaluated. But given an update operation affecting a fragment relation $\Re_j$, only the fragment constraints in *FC-i* containing $\Re_j$ in their specification should be evaluated. Since we assume that fragmentations satisfy the disjointness property, only a subset of *FC-i* is evaluated, i.e. the amount of data accessed is less than those presented in this section. Also, we assume that the worst case, i.e. $A_{FC\text{-}i} > A_{IC\text{-}i}$, is a rare case since in reality the fragmentation strategies are chosen in such a way that their effect on the integrity constraints will result in efficient constraint checking.

Constraint Distribution Techniques

The fragment constraints constructed so far involve data stored at different network sites. Because the complexity of enforcing constraints is directly related to both the number of constraints in the constraint set and the number of sites involved, our objective in this phase is to reduce the number of constraints allocated to each site for execution at that site. Distributing the whole set of fragment constraints to every site is not cost effective since not all fragment

constraints are affected by an update and so sites may not be affected by particular updates.

These techniques reduce the number of constraints allocated to each site by allocating a fragment constraint to a site if and only if there is a fragment relation at that site which is mentioned in the constraint, so that whenever an update occurs at a site, the validation of the fragment constraints at that site implies the global validity of the update. Consequently, these techniques intend to allocate each fragment constraint to a site or minimal number of sites and relieve the irrelevant sites from the computation of certain sets of fragment constraints.

The difference between our approach and Qian's approach (1989) is that the distribution techniques in our approach are applied to the end result of the constraint preprocessing techniques while the distribution techniques in Qian's approach are applied directly to the set of fragment constraints derived by her transformation rules. Our approach is more efficient than hers, since in her approach redundant processing may result during the optimization of the distributed fragment constraints. As a simple example, consider a situation where a fragment constraint requires a join between two fragment relations which are allocated to different sites of the network and this join is deduced to be empty. In our approach, this situation which is detected by the constraint preprocessing techniques will cause the fragment constraint to be eliminated from the set (i.e. no distribution is required). However, in Qian's approach this fragment constraint is distributed to both sites and the optimization of the distributed fragment constraints which is carried out at both sites will result in eliminating both distributed fragment constraints from those sites.

Table 3 shows the effect of the constraint distribution techniques on the derived sets of fragment constraints with respect to σ, which gives a rough measurement of the amount of non-local access required in constraint enforcement. As shown in the table, most of the derived sets of fragment constraints can be evaluated at a single site. Even in the worst case where each fragment is allocated to different sites of the network, the number of sites involved in evaluating the derived sets of fragment constraints is less (in most cases, generally) than the number of sites involved in evaluating the initial constraints (shown by columns Case II of Table 3).

TABLE 3
The reduction in the scatter metric

IC-i	Case I	Case II	*FC-i*	Case I	Case II
IC-1	3	3	*FC-1*	1	1
IC-2	3	3	*FC-2*	3	3
IC-3	2	2	*FC-3*	1	1
IC-4	3	5	*FC-4*	1	2
IC-5	2	2	*FC-5*	1	1
IC-6	3	5	*FC-6*	1	2

Integrity Test Generation Techniques

These techniques generate integrity tests from the syntactic structure of the constraints and the update operations. The algorithms applied for deriving these tests use the substitution, subsumption and absorption rules, and are closely related to Nicholas (1982). These algorithms can be found in Ibrahim *et al.* (1996) and are therefore omitted here.

In a distributed database, four types of integrity test can be identified. They are global post-tests, local post-tests, global pre-tests and local pre-tests (Ibrahim *et al.* 1998). We view local pre-tests as more effective in a distributed database since: (i) only a single site is involved in evaluating the local tests, i.e. the site where the update is to be performed; (ii) as they are evaluated at a target site, this avoids remote reading and the amount of data transferred across the network is minimized – in fact, no data transfer across the network is required (Gupta and Widom 1993); and (iii) they are evaluated before the update is performed – this avoids the need to undo (rollback and recover from) an update in the event of constraint violation, and this reduces the overhead cost of checking integrity (Simon and Valduriez 1986).

The integrity test generation techniques, which derive integrity tests (simplified forms) for the set of fragment constraints, further reduce the amount of data that has to be accessed, A, during the evaluation of these tests. This is illustrated by Table 5. (Table 4 presents the tests constructed for a given update operation and the set of fragment constraints, *FC-i*, given in *Fig. 2*). For example, the domain constraint *IC-1*, which initially required E amount of data to be accessed (see column $A_{IC\text{-}i}$ of Table 2), is reduced to 0 (see column A_{Ti} of Table 5), i.e. no data access is required at all; the referential integrity constraint *IC-4*, which initially required $E + D$ amount of data to be accessed (see column

TABLE 4
The test constructed for a given update operation and the set of fragment constraints, *FC-i*, given in *Fig. 2*

FC-i	INSERT	*Test,* T_i
FC-1	$emp_{2i}(a, b, c, d)$	$1.\ d > 0$
FC-2	$emp_{2i}(a, b, c, d)$	$2.\ (\forall v1 \forall v2 \forall w1 \forall w2 \forall x1 \forall y1 \forall z1)(\neg emp_1(a, v1, w1)\ V\ \neg emp_i(a, v2, w2)\ V\ \neg emp_{2i}(a, x1, y1, z1)\ V\ [(v1 = v2)\ \Lambda\ (w1 = w2)\ \Lambda\ (x1 = b)\ \Lambda\ (y1 = c)\ \Lambda\ (z1 = d)])$ *and* $(\forall x2 \forall y2 \forall z2)(\neg emp_{2j}(a, x2, y2, z2))$
	$emp_{2i}(a, b, c, d)$	$3.\ (\forall x1 \forall y1 \forall z1)(\neg emp_{2i}(a, x1, y1, z1))$
FC-3	$dept_i(a, b, c, d)$	$4.\ (\forall x1 \forall y1 \forall z1)(\neg dept_i(a, x1, y1, z1)\ V\ [(x1 = b)\ \Lambda\ (y1 = c)\ \Lambda (z1 = d)])$
	$dept_i(a, b, c, d)$	$5.\ (\forall x1 \forall y1 \forall z1)(\neg dept_i(a, x1, y1, z1))$
FC-4	$emp_{2i}(a, b, c, d)$	$6.\ (\exists x \exists y \exists z)(dept_i(b, x, y, z))$
	$emp_{2i}(a, b, c, d)$	$7.\ (\exists t \exists v \exists w)(emp_{2i}(t, b, v, w))$
FC-5	$dept_1(a, b, c, d)$	$8.\ d > 4000$
FC-6	$emp_{2i}(a, b, c, d)$	$9.\ (\forall x \forall y \forall z)(\neg dept_i(b, x, y, z)\ V\ (d \leq z))$
	$emp_{2i}(a, b, c, d)$	$10.(\exists t \exists v \exists w)(emp_{2i}(t, b, v, w)\ \Lambda\ (w \leq d))$

$A_{IC\text{-}i}$ of Table 2), is reduced to $\sum_{i=1}^{2} e_{2i} + d_i$ (see column $A_{FC\text{-}i}$ of Table 2) by the constraint preprocessing techniques and further reduced to d_i (see column A_{Ti} for test 6 of Table 5) or e_{2i} (see column A_{Ti} for test 7 of Table 5) by the integrity test generation techniques.

With respect to the scatter metric, σ, the number of sites involved in evaluating an integrity test for a local fragment constraint is 1 and the number of sites involved in evaluating an integrity test for a non-local fragment constraint is normally more than 1. A local fragment constraint is one in which all fragment relations specified in the constraint are allocated to the same site. For the case of non-local fragment constraints, the number of sites involved can be reduced to 1 by applying the integrity test generation techniques as shown in Table 5. For example, test 9 of *FC-6* for Case II (see column σ_{11} of Table 5) involves two sites, while test 10 of *FC-6* (see column σ_{11} of Table 5) for the same case involves a single site. As most of the work is being carried out at a single site, therefore the amount of data transferred across the network is minimized i.e. $\tau \approx 0$.

The integrity tests listed in Table 4 are generated from the sets of fragment constraints constructed by the fragmentation rules given in *Fig. 1.* As constraint checking is strongly influenced by the fragmentation rules used to construct the fragment relations, it is important to see and analyse the effect of applying different fragmentation rules on the derived integrity tests. This is discussed below. Assume that the fragment emp_{2i} is horizontally fragmented into two fragments emp_{2i} with some predicates different from those given in *Fig. 1*, i.e. the global relations are fragmented based on different fragmentation rules. The

TABLE 5
The reduction in the amount of data accessed, the scatter metric and the amount of data transferred across the network for the integrity tests given in Table 4

Test, T_i	A_{Ti}	σ_I [a]	σ_{II} [b]	τ_I [c]	τ_{II} [d]
1.	0	1	1	0	0
2.	$e_1 + e_1 + e_{2i} + e_{2j}$	3	3	$e_1 + e_{2j}$	$e_1 + e_{2j}$
3.	e_{2i}	1	1	0	0
4.	d_i	1	1	0	0
5.	d_i	1	1	0	0
6.	d_i	1	2	0	d_i
7.	e_{2i}	1	1	0	0
8.	0	1	1	0	0
9.	d_i	1	2	0	d_i
10.	e_{2i}	1	1	0	0

[a] σ of test T_i for Case I.
[b] σ of test T_i for Case II.
[c] τ of test T_i for Case I.
[d] τ of test T_i for Case II.

sets of fragment constraints derived after applying the constraint preprocessing techniques to their respective initial constraints are as shown in *Fig. 3*. Only the derived forms for *IC-4* (represented by $FC\text{-}4_s$) and *IC-6* (represented by $FC\text{-}6_s$) are shown, as the rest of the derived sets of fragment constraints remain the same.

$FC\text{-}4_s$: $\Lambda^2_{i=1}\mathrm{V}^2_{j=1}(\forall t\forall u\forall v\forall w\exists x\exists y\exists z)\ (emp_{2i}(t, u, v, w) \rightarrow dept_j(u, x, y, z))$
$FC\text{-}6_s$: $\Lambda^2_{i=1}\Lambda^2_{j=1}(\forall t\forall u\forall v\forall w\forall x\forall y\forall z)\ (emp_{2i}(t, u, v, w)\ \Lambda\ dept_j(u, x, y, z) \rightarrow (w \leq z))$

Fig.3: The sets of fragment constraints derived by the constraint preprocessing techniques when different fragmentation rules are applied

Table 7 shows the effectiveness of the integrity test generation techniques when applied to the sets of fragment constraints derived in *Fig. 3* with respect to A, σ and τ. (Table 6 presents the tests constructed for a given update operation and the set of fragment constraints, *FC-i*, given in *Fig. 3*). From this simple example, it is obvious that constraint checking is strongly influenced by the type of fragmentation and allocation used. Although the resulting simplified forms shown in Table 7 are not as efficient as those presented in Table 5, they are still better than the initial constraints with respect to A, σ and τ. The integrity test generation techniques formulate local simplified forms (shown by tests 12 and 14 in Table 7) which are cheaper than their alternative forms (tests 11 and 13, respectively in Table 7).

TABLE 6
The test constructed for a given update operation and the set of fragment constraints, *FC-i*, given in *Fig. 3*

FC-i	INSERT	Test, T_i
$FC\text{-}4_s$	emp_{2i}	11. $\mathrm{V}^2_{j=1}(\exists x\exists y\exists z)\ (dept_j(b, x, y, z))$
	(a, b, c, d)	12. $(\exists t\exists v\exists w)\,(emp_{2i}(t, b, v, w))$
$FC\text{-}6_s$	emp_{2i}	13. $\Lambda^2_{j=1}(\forall x\forall y\forall z)\,(dept_j(b, x, y, z)\ \mathrm{V}\ (d \leq z))$
	(a, b, c, d)	14. $(\exists t\exists v\exists w)\,(emp_{2i}(t, b, v, w)\ \Lambda\ (w \leq d))$

TABLE 7
The reduction in the amount of data accessed, the scatter metric and the amount of data transferred across the network for the integrity tests given in Table 6

Test, T_i	A_{Ti}	σ_I	σ_{II}	τ_I	τ_{II}
11.	$[d_{min}, \Sigma^2_{j=1} d_j]$	2	3	$[d_{min}, \Sigma^2_{j=1\ \text{and}\ j<>i}\ d_j]$	$[d_{min}, \sum^2_{j=1} d_j\ ]$
12.	e_{2i}	1	1	0	0
13.	$\Sigma^2_{j=1}\ d_j$	2	3	$\Sigma^2_{j=1\ \text{and}\ j<>i}\ d_j$	$\Sigma^2_{j=1}\ d_j$
14.	e_{2i}	1	1	0	0

The sufficient tests, which are normally local tests (i.e. $\sigma = 1$), are cheaper in a distributed environment (Gupta and Widom 1993; Mazumdar 1993; Qian 1989; Simon and Valduriez 1986) where the cost of accessing remote data for verifying the consistency of a database is a critical factor influencing the performance of the system (Simon and Valduriez 1986).

Tables 5 and 7 illustrate the improvement in performance gained when using our algorithm rather than Nicolas's algorithm (1982) and other techniques proposed for centralized database. In these tables, the tests 1, 2, 4, 6, 8, 9, 11 and 13 are generated by applying Nicolas's algorithm, while tests 1, 3, 5, 7, 8, 10, 12 and 14 are generated by our algorithm. Comparing those results, most of the tests generated by our algorithm are better than their alternative tests generated by Nicolas's algorithm with respect to the amount of data transferred across the network, τ and the number of sites involved, σ, i.e. the tests generated by our approach can be evaluated at a single site. This is shown in tables 5 and 7, where the tests 3, 7, 10, 12 and 14 are better than their alternative tests 2, 6, 9, 11 and 13 with respect to τ and σ, particularly for Case II.

Some conclusions can be drawn with respect to the type of constraint being considered. (I) Domain constraints (*IC-1*): the test generated will always have $A_{Ti} < A_{ICi}$ regardless of the fragmentation strategy used. In fact $A_{Ti} = 0$, $\sigma = 1$ and $\tau \approx 0$. This is because the test can be evaluated independently of the rest of the database as it only refers to the tuples to be updated (Ibrahim *et al.* 1998). (II) Key constraints (*IC-2, IC-3*): we have considered: (i) when the global relations involved in the initial constraint are fragmented on the join attribute (*IC-3*), and (ii) when the global relations involved in the initial constraint are not fragmented on the join attribute (*IC-2*). As shown in Tables 4 and 5, it is always possible to derive local tests (i.e. $\sigma = 1$) which are either (a) complete tests for case (i) (tests 4 and 5 of Table 4) or sufficient tests for case (ii) (test 3 of Table 4). These complete tests are cheaper than the initial constraint with respect to the amount of data accessed, i.e. $A_{Ti} < A_{ICi}$, σ and τ. (III) Referential constraints (*IC-4*): we have considered: (i) when the global relations involved in the initial constraint are fragmented on the reference attribute, and (ii) when the global relations involved in the initial constraint are not fragmented on the reference attribute. For both cases two types of test are generated, namely complete tests (tests 6 and 11, respectively) and sufficient tests (tests 7 and 12, respectively). The sufficient tests are cheaper than the complete tests with respect to σ and τ. (IV) General semantic integrity constraints (*IC-5, IC-6*): In general, $A_{Ti} < A_{IC_i}$ and it is always possible to generate complete tests which are normally global tests, and sometimes possible to generate tests whose $\sigma = 1$ since this depends on the fragmentation rules and the allocation schemas used, also on the complexity of the constraint itself.

In our method, given a set of possible simplified forms (integrity tests), each of the simplified forms is evaluated with respect to the three main components listed above. The most efficient one is selected based on the following heuristic rules (H1–H5). A heuristic based approach is adopted due to: (i) the difficulty

in determining all the appropriate components/parameters that can be used to measure and further select the efficient form from a range of possible simplified forms; (ii) the difficulty in assigning suitable weights to all the cost components, where the interaction between these components is not clearly defined; (iii) most of the values assigned to the components/parameters are estimates and not the actual values, which depend solely on the user input (Qian 1989); and (iv) components which are considered as critical factor that can influence the performance of a system in one situation might not be critical in other situations – for example, in a distributed database which is fully replicated, the amount of data transferred across the network in the event of evaluating constraints is not as important as in a distributed database with no replication.

Based on the above arguments, the following heuristic rules are applied:

H1 - Given a set of simplified forms, a simplified form with the lowest σ, A and τ is always preferable to the others.

H2 - As the cost of accessing remote data for verifying the consistency of a database state is the most critical factor that influences the performance of a distributed database, simplified forms which can be evaluated at a single site without involving any interaction with the remote sites are more efficient (Gupta and Widom 1993; Mazumdar 1993). Therefore, a simplified form whose $\sigma = 1$ is always preferable to the others.

H3 - In a situation where more than one local simplified form can be derived, then the simplified form with the lowest A is preferable to the others.

H4 - In a situation where the simplified forms are non-local, then the simplified form with the lowest σ is preferable to the others.

H5 - In a situation where the simplified forms are non-local with the same σ value, then the simplified form with the lowest τ is preferable to the others.

CONCLUSION

In a distributed database, the cost of accessing remote data for verifying the consistency of the database is the most critical factor that influences the performance of the system (Gupta and Widom 1993; Mazumdar 1993; Qian 1989; Simon and Valduriez 1986). In such an environment, simplified constraint forms which can be evaluated at a single site are preferable, i.e. $\sigma = 1$ and $\tau \approx 0$. In this paper, we have outlined several techniques which are essential for efficient constraint checking of fragmented relations in a distributed database. These techniques, which utilize knowledge about the database application to derive fragment constraints/simplified forms, reduce the amount of data that has to be accessed, the amount of data transferred across the network and the scatter metric which captures the scale of constraint non-locality.

Although many approaches/methods have been proposed for constructing efficient integrity tests from a given integrity constraint and its relevant update operation, these approaches/methods are mostly designed for a centralized environment. Hence, the integrity tests derived by these methods are not

suitable for a distributed environment as they often span multiple sites and involve the transfer of data across the network. In this paper, we have shown that it is always possible to generate local tests for domain, key and referential integrity constraints and it is sometimes possible to generate local tests for general semantic integrity constraints, depending on their complexity. These local tests whose evaluation only involves a single site, i.e. the site where the update is to be performed, reduce the amount of data being transferred across the network to perform integrity checks and so improve the efficiency of the integrity checking process. These local tests are derived by employing the integrity test generation techniques.

The overhead of constraint checking in a distributed environment is strongly influenced by the complexity of the integrity constraints, the fragmentation strategies used and the allocation of the fragment relations involved. In this paper, we have focused on four types of constraint, namely: domain, key, referential and simple general semantic integrity constraints. We have analyzed and demonstrated the effect of different fragmentation and allocation strategies on each of these types of constraint and shown that in most cases we gain efficiency.

There are a number of extensions and improvements that could be made: (i) Consider a broader range of constraint types. We have concentrated here on four types of constraint, namely: domain constraints, key constraints, referential integrity constraints and simple general semantic integrity constraints. Other types of constraint, such as aggregate constraints and transition constraints, are worth investigating. (ii) The overhead of constraint checking is also strongly influenced by the type of fragmentation and allocation used. Further investigation of the effect of the fragmentation and allocation strategy on the derived fragment constraints/simplified forms would be worthwhile.

REFERENCES

BARBARA, D. and H. GARCIA-MOLINA. 1992. The Demarcation Protocol: A Technique for Maintaining Linear Arithmetic Constraints in Distributed Database Systems. In *Proceedings of the Conference on Extending Database Technology* (EDBT'92). p. 373-388.

BERNSTEIN, P.A. and B.T. BLAUSTEIN. 1981. A Simplification Algorithm for Integrity Assertions and Concrete Views. In *Proceedings of the 5th International Computer Software and Applications Conference* (COMPSAC'81). p. 90-99.

DATE, C.J. 1995. *An Introduction to Database Systems.* 6th Edition. Addison-Wesley Publishing Company, Inc.

GREFEN, P.W.J. 1993. Combining Theory and Practice in Integrity Control: A Declarative Approach to the Specification of a Transaction Modification Subsystem. In *Proceedings of the 19th International Conference on Very Large Data Bases* (VLDB 19). p. 581-591.

GUPTA, A. and J. WIDOM. 1993. Local Verification of Global Integrity Constraints in Distributed Databases. In *Proceedings of the 1993 ACM SIGMOD Conference.* p. 49-58.

HENSCHEN, L.J., W.W. MCCUNE and S.A. NAQVI. 1984. Compiling Constraint-Checking Programs from First-Order Formulas. *Advances in Database Theory.* Vol. 2. Gallaire, H., Minker, J. and Nicholas, J.M. (Eds). p. 145-169. USA, New York: Plenum Press.

HSU, A. and T. IMIELINSKIN. 1985. Integrity Checking for Multiple Updates. In *Proceedings of the 1985 ACM SIGMOD International Conference on the Management of Data.* p. 152-168.

IBRAHIM, H., W.A. GRAY and N.J. FIDDIAN. 1998. Rule-Based Integrity Enforcement Strategy for a Distributed Database. In *Proceedings of the Third Biennial World Conference on Integrated Design and Process Technology – Issues and Applications of Database Technology* (IADT'98). **2**: 111-118.

IBRAHIM, H., W.A. GRAY and N.J. FIDDIAN. 1996. The Development of a Semantic Integrity Constraint Subsystem for a Distributed Database (SICSDD). In *Proceedings of the 14th British National Conference on Databases* (BNCOD 14). p. 74-91.

MAZUMDAR, S. 1993. Optimizing Distributed Integrity Constraints. In *Proceedings of the 3rd International Symposium on Database Systems for Advanced Applications.* **4**: 327-334.

MCCAROLL, N.F. 1995. Semantic integrity enforcement in parallel database machine. Ph.D Thesis. Department of Computer Science. University of Sheffield. Sheffield (UK).

MCCUNE, W.W. and L.J. HENSCHEN. 1989. Maintaining state constraints in relational databases: a proof theoretic basis. *Journal of the Association for Computing Machinery* **36(1)**: 46-68.

NICHOLAS, J.M. 1982. Logic for improving integrity checking in relational databases. *Acta Informatica* **8(3)**: 227-253.

OZSU, M.T. and P. VALDURIEZ. 1991. *Principles of Distributed Database Systems.* Prentice-Hall International Editions.

QIAN, X. 1989. Distribution Design of Integrity Constraints. In *Proceedings of the 2nd International Conference on Expert Database Systems.* p. 205-226.

QIAN, X. 1988. An Effective Method for Integrity Constraint Simplification. In *Proceedings of the 4th International Conference on Data Engineering* (ICDE 88). p. 338-345.

SIMON, E. and P. VALDURIEZ. 1986. Integrity Control in Distributed Database Systems. In *Proceedings of the 19th Hawaii International Conference on System Sciences.* p. 622-632.