A Framework for Caching Relevant Data Items for Checking Integrity Constraints of Mobile Database

doi:10.3991/ijim.v3s2.787

Zarina Dzolkhifli[1], Hamidah Ibrahim2 and Lilly Suriani Affendey[3], Praveen Madiraju[4]
[1, 2, 3] Universiti Putra Malaysia, Serdang, Selangor, Malaysia
[4] Marquette University, Milwaukee, WI, USA

***Abstract*—In a mobile environment, due to the various constraints inherited from limitations of wireless communication and mobile devices, checking for integrity constraints to maintain the consistent state of mobile databases is an important issue that needs to be addressed. Hence, in this paper we propose a framework for caching relevant data items needed during the process of checking integrity constraints of mobile databases. This is achieved by analyzing the relationships among the integrity tests (simplified form of integrity constraints) to be evaluated for a given update operation. This improves the checking mechanism by preventing delays during the process of checking constraints and performing the update. Hence, our model speeds up the checking process.**

***Index Terms*—Mobile Database, Integrity Constraints, Integrity Tests, Data Caching.**

I. INTRODUCTION

Recently, there has been an increasing interest in mobile computing due to the rapid advances in wireless communication and portable computing technologies. Massive research efforts from academia and industry have been put forth to support a new class of mobile applications such as just-in-time stock trading, mobile health services, mobile commerce, and mobile games as well as migrating the normal conventional applications to mobile applications. Users of these applications can access information at any place at any time via mobile computers and devices such as mobile phone, palmtops, laptops, and PDA [10].

While technology has been rapidly advancing, various constraints inherited from limitations of wireless communication and mobile devices remain primary challenges in the design and implementation of mobile systems and applications. These constraints include: limited client capability, limited bandwidth, weak connectivity, and user mobility. In addition, disconnections occur frequently, which may be intentional (e.g., to save battery power) or unintentional (e.g., due to signal interference). These constraints make the wireless and mobile computing environments uniquely different from a conventional wired server/client environment [10].

A general architecture of a mobile database environment is shown in Figure 1 [3, 10]. The architecture consists of base stations (BS) and mobile hosts (MH). The base station is a stationary component in the model and is responsible for a small geographic area called a cell. They are connected to each other through fixed networks. The mobile host is the mobile component of the model and may move from one cell to another. These mobile hosts communicate with the base stations through wireless networks.

Due to limited storage capabilities, a mobile host is not capable of storing all data items in the network, thus it must share some data item with a database in the fixed network. Data caching technique is used to cache some or most frequently accessed data from the base station into mobile host. By caching the needed data items, it allows mobile host to continue processing without worrying about disconnection.

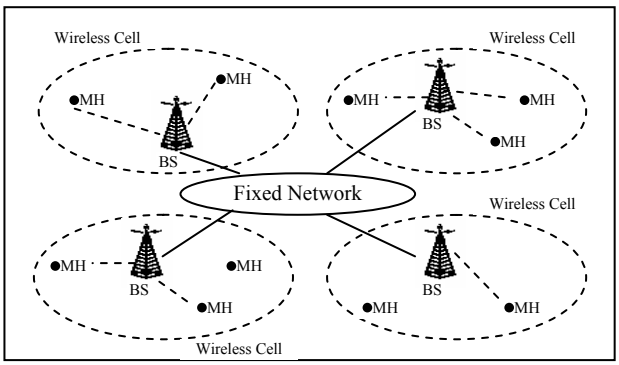

Figure 1. The architecture of a mobile database environment

Another important issue in databases is consistency, which must be maintained whenever an update operation (insert, delete, or modify) or transaction (sequence of updates) occurs at the mobile host. A database state is said to be consistent if the database satisfies a set of statements, called integrity constraints, which specify those configurations of the data that are considered semantically correct. The process of ensuring that the integrity constraints are satisfied by the database after it has been updated is termed constraint checking, which generally involves the execution of integrity tests (query that returns the value true or false). In a mobile environment, checking the integrity constraints to ensure the correctness of the database spans at least the mobile host and one other database (node), and thus the update is no longer local but rather distributed [14]. As mentioned in [14], the major problem in the mobile environment are the unbounded and unpredictable delays can affect not only the update but other updates running at both the mobile and the base stations,

which is clearly not acceptable for most applications. With the same intuition as [14], we address the challenge of extending the data consistency maintenance to cover disconnected and mobile operations.

In this paper, a framework is proposed where checking the consistency of mobile databases is performed at the mobile host. This framework is suitable for both intentional (planned) and unintentional (unplanned) disconnection. This framework differs from the approach proposed in [14] since it is intended to cater for the important and frequently used integrity constraints, i.e. those that are used in database application. Mazumdar's approach [14] is restricted to set-based constraints (equality and inequality constraints). In our work, in order not to delay the process of checking constraints during disconnection, a similar concept as proposed in distributed databases [8, 9] is employed, namely localizing integrity checking by adopting sufficient and complete tests. Since sufficient test can only verify if a constraint is satisfied, we propose that the data items required during the checking to be cached at the mobile host during the relocation period. Our approach not only treats the issue of disconnection but also reduces the amount of data items to be cached by analyzing the relationships of the integrity tests to be evaluated. Hence, we achieve speed up in the constraint checking process.

The rest of the paper is organized as follows. In Section II, the previous works related to this research are presented. In Section III, the basic definitions, notations and examples, which are used in the rest of the paper, are set out. Section IV describes the proposed framework, while conclusions are presented in the final section

II. RELATED WORK

Much of the research concerning integrity constraint checking has been conducted in the area of relational database systems. A comprehensive survey on the issues of constraint checking and maintaining in centralized, distributed and parallel databases is provided in [7]. A naïve approach is to perform the update and then check whether 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 because the evaluation of integrity constraints requires large amounts of data, which are not involved in the database update transition. Hence, improvements to this approach have been reported in many research papers. Many approaches have been proposed for constructing efficient integrity tests, for a given integrity constraint and its relevant update operation, but these approaches are mostly designed for a centralized environment [13, 16, 17]. As centralized environment has only a single site, the approaches concentrate on improving the checking mechanism by minimizing the amount of data to be accessed during the checking process. Hence, these methods are not suitable for mobile environment as the checking process often spans multiple nodes and involves the transfer of data across the network.

Several studies [1, 5, 8, 9, 11] have been conducted to improve the checking mechanism by reducing the amount of data transferred across the network in distributed databases. Nonetheless, they are not suitable for mobile databases. These approaches reformulate the global constraints into local constraints (local tests) with an implicit assumption that all sites are available, which is not true in mobile environment, where a mobile unit may be disconnected for long periods. Even though failure is considered in the distributed environment, none of the approach cater failure at the node where the update is being executed, i.e. disconnection at the target site. Nevertheless, the localization concept proposed in distributed databases is used in our approach.

Other approaches such as [6, 15] focus on the problems of checking integrity constraints in parallel databases. These approaches are not suitable for mobile databases as the intention of their approach is to speed up the checking process by performing the checking concurrently at several nodes.

To the best of our knowledge, PRO-MOTION [14] is the only work that addresses the issues of checking integrity constraints in mobile databases. The difference between our work and the work in [14] has been highlighted in the previous section.

On the other hand, to meet the characteristics of mobile devices (hosts) especially disconnection and limited storage capabilities, many previous works such as [2, 12, 18, 19, 20] have focused on strategies to cache data items into mobile host. These strategies attempt not to delay the mobile operations even during disconnection. However, these works did not focus on strategy to cache relevant data items for the purpose of checking integrity constraints at the mobile host.

III. PRELIMINARIES

Database integrity constraints are expressed in prenex conjunctive normal form with the range restricted property. 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 prefixed by $\neg$. An (in)equality is a conjunct of the form $u_1\ \theta\ u_2$ (prefixed with $\neg$ for inequality) where both u_1 and u_2 can be constants or variables and $\theta \in \{<, \leq, >, \geq, \neq, =\}$.

Integrity tests can be classified into several categories depending on the characteristics of the tests. Three different types of integrity test based on its properties were defined by McCarroll [15], namely: *sufficient tests*, *necessary tests*, and *complete tests*. An integrity test has the sufficiency property if when the test is satisfied, the associated constraint is satisfied and thus the update operation is safe with respect to the constraint. An integrity test has the necessity property if when the test is not satisfied, the associated constraint is violated and thus the update operation is unsafe with respect to the constraint. An integrity test has the completeness property if the test has both the sufficiency and the necessity properties.

Throughout this paper, the following symbols and their intended meaning, which are related to integrity constraints, are used:

- $I^{O} = \{I_1, I_2, \ldots, I_M\}$, the set of integrity constraints of an application in the whole mobile system.
- $I^{Bi} = \{I^{Bi}_1, I^{Bi}_2, \ldots, I^{Bi}_N\}$, the set of integrity constraints at the base station, i.
- $I^{Mh} = \{I^{Mh}_1, I^{Mh}_2, \ldots, I^{Mh}_O\}$, the set of integrity constraints at the mobile host, h.

From the above, $(\cup^{P}_{i=1} I^{Bi}) \cup (\cup^{Q}_{h=1} I^{Mh}) = I^{\upsilon}$, where P and Q are the number of base stations and mobile hosts, respectively in the mobile system.

Similarly, the following are the symbols and their intended meaning that are related to the data items in the mobile system. Here, data item refers to relation or fragment of relation that appears in the specification of an update operation.

- $R^{\upsilon} = \{R_1, R_2, \ldots, R_S\}$, the set of relations or fragments of relations in the mobile system.
- $R^{Bi} = \{R^{Bi}_1, R^{Bi}_2, \ldots, R^{Bi}_T\}$, the set of relations or fragments of relations at the base station, i.
- $R^{Mh} = \{R^{Mh}_1, R^{Mh}_2, \ldots, R^{Mh}_U\}$, the set of relations or fragments of relations at the mobile host, h.

From the above, $(\cup^{P}_{i=1} R^{Bi}) \cup (\cup^{Q}_{h=1} R^{Mh}) = R^{\upsilon}$, where P and Q are the number of base stations and mobile hosts, respectively in the mobile system. Also, we assume that for each data item, $R^{Mh}_v \in R^{Mh}$, the same data item appears in one of the base station, i.e. $R^{Mh}_v \in (\cup^{P}_{i=1} R^{Bi})$ [4].

Update operation in a mobile environment can occur at two different levels:

- $U^{Bi}(R)$, an update operation over the relation R, submitted by a user at the base station, i. This type of update operation is similar to the update operation in distributed databases and thus is not considered in this work. Note that R can also be a fragment of relation.
- $U^{Mh}(R)$, an update operation over the relation R, submitted by a user through his mobile host, h, where R is located at the mobile host. Note that R can also be a fragment of relation.

Throughout this paper the *company* database is used, as given in Figure 2. Table 1 presents some of the integrity tests generated based on the set of integrity constraints given in Figure 2. The derivation of the integrity tests is omitted here since this is not the focus of this paper. Interested readers may refer to [8, 9].

Schema:
emp(eno, dno, ejob, esal);
dept(dno, dname, mgrno, mgrsal);
proj(eno, dno, pno)
Integrity Constraints:
'A specification of valid salary'
I_1: $(\forall w \forall x \forall y \forall z)(emp(w, x, y, z) \rightarrow (z > 0))$
'Every employee has a unique *eno*'
I_2: $(\forall w \forall x1 \forall x2 \forall y1 \forall y2 \forall z1 \forall z2)(emp(w, x1, y1, z1) \wedge emp(w, x2, y2, z2) \rightarrow (x1 = x2) \wedge (y1 = y2) \wedge (z1 = z2))$
'Every department has a unique *dno*'
I_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'
I_4: $(\forall t \forall u \forall v \forall w \exists x \exists y \exists z)(emp(t, u, v, w) \rightarrow dept(u, x, y, z))$
'The *eno* of every tuple in the *proj* relation exists in the *emp* relation'
I_5: $(\forall u \forall v \forall w \exists x \exists y \exists z)(proj(u, v, w) \rightarrow emp(u, x, y, z))$
'The *dno* of every tuple in the *proj* relation exists in the *dept* relation'
I_6: $(\forall u \forall v \forall w \exists x \exists y \exists z)(proj(u, v, w) \rightarrow dept(v, x, y, z))$
'Every manager in *dept* '*D*1' earns > £4000'
I_7: $(\forall w \forall x \forall y \forall z)(dept(w, x, y, z) \wedge (w = \text{'}D1\text{'}) \rightarrow (z > 4000))$
'Every employee must earn ≤ to the manager in the same department'
I_8: $(\forall t \forall u \forall v \forall w \forall x \forall y \forall z)(emp(t, u, v, w) \wedge dept(u, x, y, z) \rightarrow (w \leq z))$
'Any department that is working on a project P_1 is also working on project P_2'
I_9: $(\forall x \forall y \exists z)(proj(x, y, P_1) \rightarrow proj(z, y, P_2))$

Figure 2. The Company static integrity constraints

TABLE I.
THE INTEGRITY TESTS DERIVED BASED ON THE INTEGRITY CONSTRAINTS LISTED IN FIGURE 2

I^{υ}	Update Template	Integrity Test
I_1	insert($emp(a, b, c, d)$)	1. $d > 0$[1]
I_2	insert($emp(a, b, c, d)$)	2. $(\forall x2 \forall y2 \forall z2)(\neg emp(a, x2, y2, z2) \vee [(b = x2) \wedge (c = y2) \wedge (d = z2)])$ [1]
I_3	insert($dept(a, b, c, d)$)	3. $(\forall x2 \forall y2 \forall z2)(\neg dept(a, x2, y2, z2) \vee [(b = x2) \wedge (c = y2) \wedge (d = z2)])$[1]
I_4	insert($emp(a, b, c, d)$)	4. $(\exists x \exists y \exists z)(dept(b, x, y, z))$[1]
		5. $(\exists t \exists v \exists w)(emp(t, b, v, w))$[2]
	delete($dept(a, b, c, d)$)	6. $(\forall t \forall v \forall w)(\neg emp(t, a, v, w))$[1]
I_5	insert($proj(a, b, c)$)	7. $(\exists x \exists y \exists z)(emp(a, x, y, z))$[1]
		8. $(\exists v \exists w)(proj(a, v, w))$[2]
	delete($emp(a, b, c, d)$)	9. $(\forall v \forall w)(\neg proj(a, v, w))$[1]
I_6	insert($proj(a, b, c)$)	10. $(\exists x \exists y \exists z)(dept(b, x, y, z))$[1]
		11. $(\exists u \exists w)(proj(u, b, w))$[2]
	delete($dept(a, b, c, d)$)	12. $(\forall u \forall w)(\neg proj(u, a, w))$[1]
I_7	insert($dept(a, b, c, d)$)	13. $(a \neq \text{'}D1\text{'}) \vee (d > 4000)$[1]
I_8	insert($emp(a, b, c, d)$)	14. $(\forall x \forall y \forall z)(\neg dept(b, x, y, z) \vee (d \leq z))$[1]
		15. $(\exists t \exists v \exists w)(emp(t, b, v, w) \wedge (w \geq d))$[2]
I_9	insert($proj(a, b, P1)$)	16. $(\exists z)(proj(z, b, P2))$[1]
		17. $(\exists z)(proj(z, b, P1))$[2]
	delete($proj(a, b, P2)$)	18. $(\forall x)(\neg proj(x, b, P1))$[1]
		19. $(\exists z)(proj(z, b, P2) \wedge (z \neq a))$[2]

Note: a, b, c and d are generic constants; [1]: complete test; and [2]: sufficient test.

IV. THE PROPOSED FRAMEWORK

The proposed framework is illustrated in Figure 3. The framework consists of 4 main components. These components are:

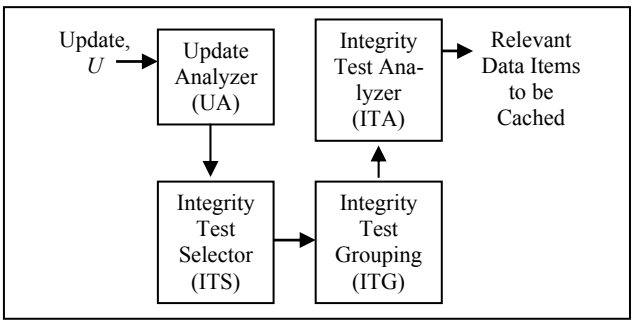

Figure 3. The proposed framework

(a) Update Analyzer (UA): This component accepts an update operation submitted by a user, $U^{Mh}(R)$, and analy-

ses the operation to identify the type of update operation (insert, delete, modify), the relation involved, and the set of data values to be inserted/deleted/modified**.**

(b) Integrity Test Selector (ITS): When a user requests an update, only those constraints that might be violated are selected for evaluation. Based on these constraints, the appropriate tests are selected. Thus, this component selects the integrity tests to be triggered, by comparing the type of update operation and the relation of the user's update operation with the type of update operation and relation of each of the update template stored in the mobile host. For example, if $I^{Mh} = \{I_1, I_2, I_4, I_5, I_8\}$ and $U^{Mh}(R)$ = insert(*emp*(*E*20, *D*1, Analysts, 3400)), then tests 1, 2, 4, 5, 14, and 15 are selected.

(c) Integrity Test Grouping (ITG): This component groups the integrity tests that have been selected by ITS. There are several criteria that can be used for this purpose. For example grouping can be based on the relation specified in the tests, i.e. those tests that will be evaluated over the same relation are grouped together in the same group. Grouping can also be based on the type of tests, i.e. those tests that have the same properties (complete or sufficient) are assigned to the same group. Other criterion that can be used is region, i.e. tests that can be evaluated locally are grouped in the same group. Note also that it is seldom the case that we can select a test from each of the integrity constraint that satisfies the characteristics of the group. Thus, a group may have some tests whose characteristics do not belong to the group but are forced to be the elements of the group, since they are the only tests available for a given integrity constraint. After grouping, this component is also responsible to select one of the group to be evaluated. For the example given in (b) above, the following are some possible groups:

Based on relation (Note, *Gi* is a label for a group):

*G*1: {1, 2, 5, 15}

All tests span the *emp* relation except test 1.

*G*2: {1, 2, 4, 14}

All tests span the *dept* relation except tests 1 and 2. Tests 1 and 2 are selected and grouped in *G*2 as they are the only tests available for I_1 and I_2, respectively.

Based on properties of the tests:

*G*3: {1, 2, 4, 14}

All tests are complete tests.

*G*4: {1, 2, 5, 15}

All tests are sufficient tests except tests 1 and 2. Tests 1 and 2 are selected and grouped in *G*4 as they are the only tests available for I_1 and I_2, respectively.

Based on region: Assume that only part of the *emp* relation is located at the mobile host.

*G*5: {1, 2, 5, 15}

Test 1 is a local test, while tests 2, 5, and 15 have high chances to be evaluated locally.

*G*6: {1, 2, 4, 14}

Test 1 is a local test, test 2 has high chances to be evaluated locally, while tests 4 and 14 are global tests.

Finally, one of these groups is selected to be evaluated. Decision to select is based on the data items already located at the mobile host. If the tests of the group have more chances to be performed locally at the mobile host, then that group is selected.

(d) Integrity Test Analyzer (ITA): This is the core component of the whole framework that analyses the relationships among the tests that have been grouped and selected to be evaluated by ITG, with the aim to identify the relevant data items to be cached. Here, relevant is defined as the minimum number of data items that needs to be cached given a set of integrity tests to be evaluated. Analysis is performed by comparing the relations, constant values, and equations among the tests. Three main rules are applied as follows:

Rule 1: Test T_i is said to be *redundant* with test T_j if the data item(s) required by both T_i and T_j is the same, i.e. $D_i \cap D_j = D_i$ where D_i and D_j denote the set of data items needed by T_i and T_j, respectively.

Rule 2: Test T_i is said to be *subsumed* by test T_j if the data item(s) required by T_i is part of the data item(s) required by T_j, i.e. $D_i \subseteq D_j$.

Rule 3: Test T_i is said to be *contradicted* with test T_j if the data item(s) required by T_i is not the data item(s) required by T_j although the attribute(s) is the same, i.e. $D_i \cap D_j = \{\}$ and both D_i and D_j are over the same attribute.

The steps performed at this stage are as follows:

1. BEGIN
2. Substitute each test in the group *Gi* with the actual values as given in the update operation, $U^{Mh}(R)$.
3. Evaluate domain test (if any). If the test is false, then $U^{Mh}(R)$ is aborted. GO TO step 7.
4. For each of the remaining tests in the *Gi*, identify the data items required by the test.
5. Check for redundancy, subsumption, and contradiction by applying rules 1, 2, and 3.
6. Generate the required relevant data items needed to be cached from base station. For Rule 1, D_i is cached. While for Rule 2, D_j is cached and for Rule 3, both D_i and D_j are cached.
7. END

For example, assume that *G*1 has been selected.

Example 1:

Step 2:

1. $3400 > 0$
2. $(\forall x2 \forall y2 \forall z2)(\neg emp(E20, x2, y2, z2) \vee [(D1 = x2) \wedge (Analysts = y2) \wedge (3400 = z2)])$
5. $(\exists t \exists v \exists w)(emp(t, D1, v, w))$
15. $(\exists t \exists v \exists w)(emp(t, D1, v, w) \wedge (w \geq 3400))$

Step 3:

1. $3400 > 0$ is true.
2. $(\forall x2 \forall y2 \forall z2)(\neg emp(E20, x2, y2, z2) \vee [(D1 = x2) \wedge (Analysts = y2) \wedge (3400 = z2)])$
5. $(\exists t \exists v \exists w)(emp(t, D1, v, w))$
15. $(\exists t \exists v \exists w)(emp(t, D1, v, w) \wedge (w \geq 3400))$

Step 4:

Test	Relation	Attribute	Value
2	*emp*	*eno*	*E*20
		dno	*D*1
		ejob	*Analysts*
		esal	3400
5	*emp*	*dno*	*D*1
15	*emp*	*dno*	*D*1
		esal	≥ 3400

Step 5:

The data item needed by test 5 is part of the data items required by test 15 (Rule 2).

Step 6:

Thus, the data items to be cached (if and only if the data items are not at the mobile host) are as follows:

Relation	Attribute	Value
emp	*eno*	*E*20
emp	*dno*	*D*1
	esal	≥ 3400

As for a second example, consider $I^{Mh} = \{I_5, I_6, I_9\}$ and $U^{Mh}(R)$ = insert(*proj*(*E*20, *D*1, *P*1)), then tests 7, 8, 10, 11, 16, and 17 are selected. Assume that the following group has been selected by ITG, $G7 = \{7, 10, 16\}$ (complete tests).

Example 2:

Steps 2 and 3:

7. $(\exists x \exists y \exists z)(emp(E20, x, y, z))$
10. $(\exists x \exists y \exists z)(dept(D1, x, y, z))$
16. $(\exists z)(proj(z, D1, P2))$

Step 4:

Test	Relation	Attribute	Value
7.	*emp*	*eno*	*E*20
10.	*dept*	*dno*	*D*1
16.	*proj*	*dno*	*D*1
		pno	*P*2

Step 5:

The data item needed by test 10 is part of the data items required by test 16 (Rule 2).

Step 6:

Relation	Attribute	Value
emp	*eno*	*E*20
proj	*dno*	*D*1
	pno	*P*2

We have performed a simple analysis that compares (a) caching the whole data item without analyzing the integrity tests, (b) caching the data items by analyzing the integrity tests individually (i.e. omitting Step 5), and (c) caching the data items by analyzing the relationships between the integrity tests. For this analysis, we assume the following, the *emp* relation has 500 tuples (2000 data items), *dept* has 10 tuples (40 data items), and *proj* has 100 tuples (300 data items). Note the number of data items is calculated by multiplying the number of tuples with the number of attributes of a relation. Figure 4 illustrates this comparison. From this figure, we can conclude that the number of data items to be cached can be significantly reduced by analyzing the relationships among the integrity tests. Another analysis has been conducted by increasing the number of tuples (records) in each relation. The results are as shown in Figure 5. From Figure 5 we noticed that increasing the number of tuples in each relation has no effect on the number of data items to be cached for both strategies (b) and (c).

The proposed framework improves the constraint checking mechanism mainly by employing an efficient checking strategy, which is achieved through:

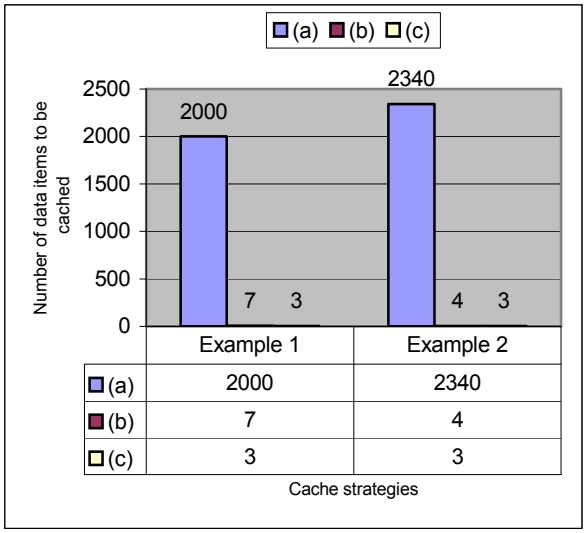

Figure 4: Comparison between strategies (a), (b), and (c), with respect to the number of data items to be cached

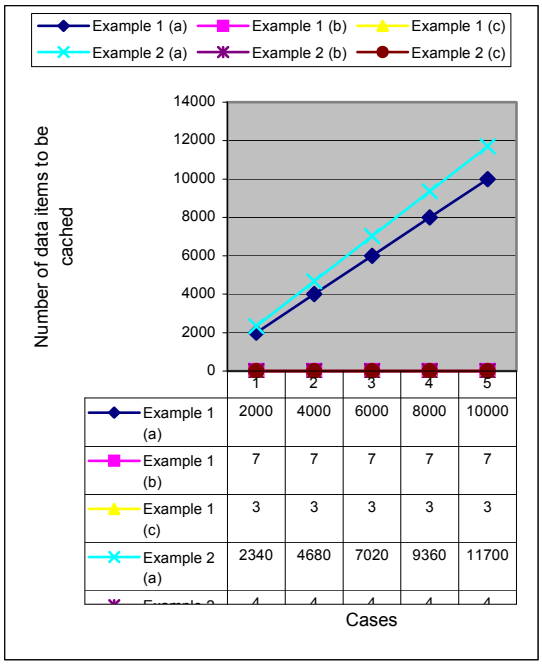

Figure5: Comparison between strategies (a), (b), and (c), when the number of tuples in each relation is increased

(i) Caching relevant data items – this has achieved two purposes, namely: (i) upgrading the properties of the tests – by caching the relevant data items it increases the possibility of performing the constraints checking locally at the mobile host, as most of the data required are now available at the mobile host and (ii) the process of checking the integrity constraints at the mobile host can be performed without delay even if the mobile host is disconnected.

(ii) Localizing integrity checking – allow the initial constraints to be validated by accessing data at the mobile host, i.e. at the site where the update is performed. This technique eliminates the cost of accessing remote data, i.e. it minimizes inter-site data communication cost. It also prevents delays during the process of checking constraints and performing the update, especially when the mobile host is disconnected.

(iii) Test filtering – for each update request, only those constraints that may be violated by it are selected for further evaluation.

V. CONCLUSION

This paper has presented a framework, which is designed for checking database integrity in a mobile environment. This framework adopts the simplified forms of integrity constraints, namely: sufficient and complete tests, together with the idea of caching the relevant data items during the relocation period for the purpose of checking the integrity constraints. It has improved the performance of the checking mechanism of mobile databases as delay during the process of checking the integrity constraints and performing the update is reduced.

REFERENCES

[1] Alwan, A.A., Ibrahim, H., and Udzir, N.I., "Local Integrity Checking using Local Information in a Distributed Database", *Proceedings of the 1st Aalborg University IEEE Student Paper Contest 2007 (AISPC'07)*, Aalborg, 2007.

[2] Chan, B.Y., Si, A., and Leong, H.V., "A Framework for Cache Management for Mobile Databases: Design and Evaluation", *Distributed and Parallel Databases*, Vol. 10, 2001, pp. 23-57. (doi:10.1023/A:1019297705159)

[3] Chan, D. and Roddick, J.F., "Context-sensitive Mobile Database Summarization", *Proceedings of the Twenty-Sixth Australian Computer Science Conference (ACSC 2003)*, Adelaide, 2003.

[4] EPFL, Grenoble, U., INRIA-Nancy, INT-Evry, Montpellier, U., Paris, U., and Versailles, U., "Mobile Databases: a Selection of Open Issues and Research Directions", *SIGMOD Record*, Vol. 33, No. 2, 2004, pp. 78-83.

[5] Gupta, A., "Partial Information Based Integrity Constraint Checking", PhD Thesis, Stanford University, USA, 1994.

[6] Hanandeh, F.A.H., "Integrity Constraints Maintenance for Parallel Databases", PhD Thesis, UPM, Malaysia, 2006.

[7] Ibrahim, H., "Checking Integrity Constraints – How it Differs in Centralized, Distributed and Parallel Databases", *Proceedings of the 17th International Conference on Database and Expert Systems Applications – the Second International Workshop on Logical Aspects and Applications of Integrity Constraints (LAAIC'06)*, Krakow, 2006, pp. 563-568.

[8] Ibrahim, H., "A Strategy for Semantic Integrity Checking in Distributed Databases", *Proceedings of the Ninth International Conference on Parallel and Distributed Systems*, IEEE Computer Society, Republic of China, 2002.

[9] Ibrahim, H., Gray, W.A., and Fiddian, N.J., "Optimizing Fragment Constraints – A Performance Evaluation", *International Journal of Intelligent Systems – Verification and Validation Issues in Databases, Knowledge-Based Systems, and Ontologies*, Edited by: Ronald, R., John Wiley & Sons Inc., Vol. 16, No. 3, 2001, pp. 285-306.

[10] Ken, C.K.L, Wang-Chien, L., and Sanjay, M., "Pervasive Data Access in Wireless and Mobile Computing Environments", *Journal of Wireless Communications and Mobile Computing*, 2006.

[11] Madiraju, P. and Sunderraman, R., "A Mobile Agent Approach for Global Database Constraint Checking", *Proceedings of the ACM Symposium on Applied Computing (SAC'04)*, Nicosia, 2004, pp. 679-683.

[12] Madria, S.K., Mohania, M., Bhowmick, S.S., and Bhargava, B., "Mobile Data and Transaction Management, Information Sciences", *Elsevier*, 2002, pp. 279-309.

[13] Martinenghi, D., "Advanced Techniques for Efficient Data Integrity Checking", *PhD Thesis*, Roskilde University, 2005.

[14] Mazumdar. S. and Chrysanthis, P.K., "Localization of Integrity Constraints in Mobile Databases and Specification in PROMOTION", *Proceedings of the Mobile Networks and Applications*, 2004, pp. 481-490.

[15] McCarroll, N.F., "Semantic Integrity Enforcement in Parallel Database Machines", *PhD Thesis*, University of Sheffield, UK, 1995.

[16] McCune, W.W. and Henschen, L.J., "Maintaining State Constraints in Relational Databases: a Proof Theoretic Basis", *Journal of the Association for Computing Machinery*, Vol. 36, No. 1, 1989, pp. 46-68.

[17] Nicolas, J.M., "Logic for Improving Integrity Checking in Relational Data Bases", *Acta Informatica*, Vol. 18, No. 3, 1982, pp. 227-253. (doi:10.1007/BF00263192)

[18] Pitoura, E. and Chrysanthis, P.K., "Caching and Replication in Mobile Data Management", *IEEE Data Engineering*, Bull, 2007, pp. 13-20.

[19] Ren, Q., Dunham, H.M., and Kumar, V., "Semantic Caching and Query Processing", *IEEE Transaction on Knowledge and Data Engineering*, Vol. 15, 2003, pp. 192-210. (doi:10.1109/TKDE.2003.1161590)

[20] Song, H. and Cao, G., "Cache-miss-initiated Prefetch in Mobile Environment", *Science Direct*, *Computer Communication*, Vol. 28, 2005, pp. 741-753.

AUTHORS

Zarina Dzolkhifli, is a postgraduate student of Department of Computer Science, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 Serdang, Selangor, Malaysia. (e-mail: zarinadzolkhifli@yahoo.com.sg).

Hamidah Ibrahim, is an associate professor of the Department of Computer Science, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 Serdang, Selangor, Malaysia. (e-mail: hamidah@upm.edu.my).

Lilly Suriani Affendey, is a senior lecturer of the Department of Computer Science, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 Serdang, Selangor, Malaysia. (e-mail: suriani@fsktm.upm.edu.my).

Praveen Madiraju, is an assistant professor of the Department of Mathematics, Statistics and Computer Science, Marquette University, Milwaukee WI 53201-1881 USA. (e-mail: praveen@mscs.mu.edu).

This work was supported by the Malaysian Ministry of Science, Technology and Innovation (MOSTI) under grant number 01-01-04-SF0340.

Submitted 24 December 2008. Published as resubmitted by the authors on 9 October 2009.