Caching Constrained Mobile Data *

Subhasish Mazumdar
Computer Science Dept.
New Mexico Tech
Socorro, NM 87801
mazumdar@nmt.edu

Mateusz Pietrzvk [†]
Computer Sciencebept.
New Mexico Tech
Socorro, NM 87801
mateuszp@quidnunc.com

Panos K. Chrvsanthis
Computer Science Dept.
University of Pittsburgh
Pittsburgh, PA 15260
panos@cs.pitt.edu

ABSTRACT

As mobile devices get ubiquitous and grow in computational power, their management of interdependent data also becomes increasingly important. The mobile environment exhibits all the characteristics of a distributed database plus the feature of whimsical connectivity. Consequently, transactions respecting data consistency can suffer unbounded and unpredictable delays at both mobile and stationary nodes. The currently popular multi-tier model, in which mobile devices are in one end and always-connected stationary servers in the other, has certain practical advantages. However, it assumes that all integrity constraints are evaluated at the servers and hence relies on the semantics of operations for any autonomy enhancement of the mobile devices. In this paper, we examine the idea of constraint localization in cases where two mobile nodes each own data that share a constraint. It relies on reformulation of a constraint into more flexible local constraints that give more autonomy to the mobile nodes. The scheme also involves dynamic changes of these local constraints through negotiation, which we call re-localization. To overcome the problem of simultaneous requests for such re-localization, we give algorithms along with experimental results indicating their effectiveness.

1. INTRODUCTION

As mobile devices increase in popularity and grow stronger in computational power, their role as a seamless extension of databases on stationary servers becomes increasingly attractive. This means that mobile databases need transaction support; however, the standard client-server database technology is not quite directly applicable. The most important and interesting difference is that mobile devices when disconnected, are often operational. Disconnection from the fixed network can be either involuntary, caused by the nature of the physical environment around the mobile device, or voluntary, owing to the fragility of battery power and the economics of wireless communication. Consequently, information stored within the mobile device becomes crucial to maintaining productivity during a period of disconnection.

*This work was partially supported by NSF awards IRI-9509789, IRI-95020091, and IIS-9812532.
†Currently at Quidnunc Inc., San Francisco, CA.

Client data caching and dynamic data replication, in particular, improve both performance as well as data availability [1]. In order to be more effective, the cache granularity should be that of objects instead of pages, with an inter-transactional lifetime, i.e., they are potentially accessed by more than one transaction. Further, to preserve correctness, we require a cache consistency protocol [3] that takes into account the whimsical connectivity of mobile nodes. Towards this end, many cache consistency algorithms have been proposed supporting different degrees of consistency from *strict* (eager replication) to *eventual* (lazy replication) consistency. In eager, all replicas are updated atomically. In lazy, replica updates are asynchronously propagated to other nodes after the updating transaction commits; while this is useful for disconnection, inconsistencies that may need (typically manual) reconciliation. In [4], the *two-tier* replication model was proposed as a practical compromise between the two.

Motivation

The two-tier replication model, generalized to the multi-tier model has become popular because it allows mobile nodes to read and update replicated objects during disconnections while it avoids concurrency anomalies such as loss of updates. Our goal in this paper is to extend its applicability and effectiveness to more transaction classes than the ones considered in the original model.

In the two-tier replication scheme, each object is mastered at one node: a few by mobile nodes, most by stationary or base nodes. A mobile node has a copy of an object mastered at a base node and creates a tentative version when it updates it. There are two kinds of transactions: a base transaction runs on one or more base nodes and at most one connected mobile node, accessing only master data and producing master data. Base transactions execute under one-copy serializability so the master state is a result of an execution equivalent to some serial execution using no replication. A tentative transaction T_1 originates at a mobile node, accesses data mastered on that node and possibly copies of objects mastered at base nodes and produces tentative versions; it is later run as a base transaction accessing all master data when the mobile node re-connects with a base node. It can now fail because of a constraint violation; such failure can be made less likely through semantic tricks, e.g., commutative updates.

In general, however, the situation is more difficult. For example, suppose D_0 is a data item mastered on a base node B_0 while D_1, D_2, and D_3 are mastered on mobile nodes M_1, M_2, and M_3, respectively. Also suppose $p(D_0, D_1)$ and $q(D_2, D_3)$ are integrity constraints. Now consider transactions T_0 and T_2:

- T_0 runs on base node B_0 and updates D_0. This transaction cannot commit until M_1 is connected and the master D_1 is checked for constraint violation.
- T_2 at M_2 updates D_2. It too can only commit tentatively. However, even with M_2 connected, T_2 cannot be re-run as a base transaction until M_3 is connected.

Transactions T_0 and T_2 may suffer inordinate delays. Perhaps for this reason, such transactions were outlawed in the two-tier replication model.

Approach and Contribution

In this paper, we use the idea of constraint localization to allow transactions such as T_0 and T_2 to be accepted within the two-tier replication model while eliminating the probability of messy reconciliation for a tentative transaction T_1.

Our approach based on constraint localization is a preemptive one. In the above scenario, the global constraint will be reformulated into local constraints by a process we call localization. The key idea is that for some distributed constraints, it is possible to find a conjunction of entirely local constraints that forms a *sufficient* condition for the original constraint [8]. Such a local constraint could very well be more restrictive than the original one: that is the cost while the benefit is the enhanced autonomy so obtained. As a reflection of this autonomy, T_0, T_1, T_2 all shed their constraint violation tentativeness. The theoretical framework behind our approach for mobile databases is sketched in [9].

Given that local constraints could be more restrictive, it becomes necessary to readjust the local constraints dynamically on demand; we refer to this as *relocalization.* It would work as follows. Let constraint p be reformulated into local constraints p_0 on D_0 (local to B_0) and p_1 on D_1 (local to M_1) and similarly q into local constraints q_2 on D_2 (local to M_2) and q_3 on D_3 (local to M_3). Suppose that a transaction on mobile node M_1 fails the local constraint p_1 on data D_1; since p_1 is a sufficient condition only, it is possible that a dynamic adjustment of the sufficient conditions on D_0 and D_1 may yield a more favorable p_1 at M_1. To negotiate this, M_1 contacts a base node B. Note that we have traded the problem of tentative commits (that may fail later and require reconciliation) for a new problem of aborts (that may succeed later with a readjustment of constraints); the tradeoff is not symmetric because we avoid the messy problem of reconciliation. In this example, B having full access to D_0, performs this negotiation. In this paper, we propose to achieve this in the general case using strong local constraints managed by constraint guardians. A constraint guardian is a process that runs on the stationary network and is assigned to a global integrity constraint involving mastered data, at least one of which is mastered by a mobile node. In some sense, constraint guardians are metadata proxy managers [13] for mobile mastered data. The complexity of a constraint guardian depends on the type of constraint it is responsible for.

For integrity constraints, we consider linear and quadratic inequalities. Linear inequalities are common in traditional databases; support for quadratic inequalities became important in emerging applications [14] where quality assurance is essential, e.g., in the trucking industry which we were studying [9]. We present a geometric method that works for both classes and is the basis of the initial localization and dynamic relocalization performed by the constraint guardian. Relocalization must deal with the problem of constraint change requests from more than one node simultaneously. For this purpose, we have developed, implemented and tested new algorithms; we discuss them as well as the results obtained.

In the next section, we explain localization. In the following section, we explore how it can be used to expand the applicability of the two-tier replication model and introduce the notion of constraint guardians. Finally, we outline the relocalization algorithm for handling simultaneous requests for constraint change along with experimental results.

2. USING LOCALIZATION

2.1 An Example

The trucking industry is increasingly using mobile computers [17]. Each truck is equipped with a computer and a satellite or radio link. We envision a truck picking up goods while also checking to see if they meet their specifications, i.e., if a quality control attribute A (e.g., diameter of a washer) is within its specified tolerance. The idea is to perform quality control during pick-up itself so as to avoid returning unsatisfactory goods later.

In our example, two trucks independently are assigned to pick up Q_1, Q_2 number of goods respectively. However, these are partial contracts. When the two truckloads are merged at the destination, the overall mean μ and variance σ^2 of A of *the merged* collection must be within tolerable limits: $M_0 \leq \mu \leq M_1$ and $0 \leq \sigma^2/\mu^2 \leq K$, where M_0, M_1, K are constants (we capitalize constants). Each truck's mobile computer measures the mean and variance of A for the goods it picks up. If these two metrics are outside their acceptable range, the goods are rejected on the spot.

Assume that the quality control attribute A is uniformly distributed at the two sources. Let the two trucks observe means μ_1, μ_2 and variances σ_1^2, σ_2^2 while handling quantities Q_1 and Q_2 respectively. The fraction of goods handled by them are $R_1 = Q_1/(Q_1 + Q_2)$, $R_2 = (1 - R_1)$ respectively. Then, the restriction on the overall μ and σ^2 lead to constraints P1 through P4 (two linear and two quadratic polynomial inequalities in four variables μ_1, μ_2, σ_1^2, σ_2^2):

$$
\begin{aligned}
&\text{P1: } R_1\mu_1 + R_2\mu_2 \geq M_0 \\
&\text{P2: } R_1\mu_1 + R_2\mu_2 \leq M_1 \\
&\text{P3: } R_1\sigma_1^2 + R_2\sigma_2^2 + R_1 R_2\mu_1^2 + R_1 R_2\mu_2^2 - 2R_1 R_2\mu_1\mu_2 \geq 0 \\
&\text{P4: } R_1\sigma_1^2 + R_2\sigma_2^2 + R_1(1 - R_1 - KR_1)\mu_1^2 \\
&\qquad + R_2(R_1 - KR_2)\mu_2^2 + R_2(R_1 - KR_2)\mu_2^2 \\
&\qquad - 2R_1R_2(1 + K)\mu_1\mu_2 \leq 0
\end{aligned}
$$

Clearly, it makes sense to master μ_1, σ_1^2 at the first truck instead of the stationary node (similarly μ_2, σ_2^2 at the second truck). But the truck driver after measuring μ_1, σ_1^2, must attempt to verify P1 through P4. But to do so, he/she must access μ_2, σ_2^2 measured by the second truck, thus violating the precondition of the two-tier model. We will show in the next section that localization solves this problem.

2.2 Formalizing the Notion

We assume that data is distributed among nodes $1, 2, \ldots, N$. We call a constraint C *local* if it involves only one node and *distributed* otherwise.

Definition 1: A distributed constraint $C(\overline{x_1}, \ldots, \overline{x_N})$ where $\overline{x_i}$ reside in node i $(1 \leq i \leq N)$ is said to be localizable if there is a **non-trivial** rule

$$C_1(\overline{x_1}) \wedge C_2(\overline{x_2}) \wedge \ldots \wedge C_N(\overline{x_N}) \rightarrow C(\overline{x_1}, \ldots, \overline{x_N}),$$

such that C_i is local for $1 \leq i \leq N$.

The variables and quantifiers (not shown) conform to the rules for Horn clauses [7]. The rule is trivial if any C_i in the left hand side (LHS) is false. We denote the LHS of the rule by SC, a *sufficient* condition for C, and say that C is localizable *through* SC (or that SC localizes C). So, instead of enforcing C which is distributed, we enforce the local constraint C_i at each node i for $1 \leq i \leq N$.

For example, let C be **P1**. It is a distributed constraint involving 2 nodes (the 2 trucks), i.e., N = 2; it uses variables μ_1 and μ_2 respectively. Using the rule $C_1 \wedge C_2 \rightarrow P1$, where $C_1 = (\mu_1 \geq L_1)$, and $C_2 = (\mu_2 \geq L_2)$, and L_1, L_2 are constants such that $R_1L_1 + R_2L_2 \geq M_0$, we see that P1 is localizable. Thus, we can enforce $\mu_1 \geq L_1$ at the first truck and $\mu_2 \geq L_2$ at the second, both local constraints, assured that their simultaneous enforcement implies P1.

Now, since SC is only sufficient for C, a local update may violate C_i and hence SC, while still satisfying C. In this case, we would like SC to be dynamically transformable to, say, SC' that could accommodate the updated value. SC' would be of the form:

$$SC' = C'_1 \wedge C'_2 \wedge \ldots \wedge C'_N.$$

Though the transformation of SC to SC' typically involves constraint changes at more than one node, we want to achieve this in an incremental node-by-node manner (perhaps in a pre-determined order), avoiding synchronization delays due to commit protocols arising from distributed transactions.

Definition 2: SC is said to be *relocalizable* to SC' *through a* sequence of constraints $W_0(= SC), \ldots, W_i, \ldots, W_N$ (= SC') if $W_i \rightarrow$ C for $0 \leq i \leq N$ and the sequence is incremental in the sense that W_{i-1} and W_i differ in only one conjunct C_j (for some j) in W_{i-1} which gets replaced by C'_j in W_i. Such a transformation of SC to SC' is referred to as relocalization.

Returning to our example, if the first truck observes a mean $\mu_1 < L_1$, it does not mean that P1 is violated. It may be possible to reduce L_1 to L'_1 and increase L_2 to L'_2 such that $R_1L_1 + R_2L_2 = R_1L'_1 + R_2L'_2$; also, L_1, L_2 *can* be changed without a distributed transaction if the second truck increases L_2 before the first decreases L_1. This is relocalization through a sequence W_0, W_1, W_2 where

$$\begin{aligned} W_0 &= SC = (\mu_1 \geq L_1) \wedge (\mu_2 \geq L_2), \\ W_1 &= (\mu_1 \geq L_1) \wedge (\mu_2 \geq L'_2), \text{ and} \\ W_2 &= SC' = (\mu_1 \geq L'_1) \wedge (\mu_2 \geq L'_2). \end{aligned}$$

W_0, W_1, W_2 all imply the original constraint P1, i.e., at each step, the original constraint P1 is maintained. If the order of constraint changes was reversed (i.e., the fist truck decreased L_1 before the second truck increased L_2), the resulting intermediate state with $W_1 = (\mu_1 \geq L'_1) \wedge (\mu_2 \geq L_2)$ may not have satisfied P1.

Note that this is basically what is achieved by the Escrow method [5, 6, 16] and Demarcation Protocol [2], but these methods neither work for constraints P3 and P4 (owing to the product term $\mu_1\mu_2$, P3 and P4 cannot be converted into a linear form), nor can these methods be easily extended to do so. We will show in the next subsection that our method, by exploiting the localization perspective, works uniformly for constraints P1 through P4. In general, our method works for distributed polynomial inequality constraints.

The following remarks cover some useful properties of localization and indicate why node autonomy is enhanced.

Rm1 If a local transaction at node i satisfies the local constraint C_i, no global constraint needs to be checked and therefore unpredictable delays are avoided.

Rm2 Node i can unilaterally change its local constraint from C_i to C'_i if the new constraint is more restrictive, i.e., $C'_i \rightarrow C_i$, provided the data which now satisfies C_i will also satisfy C'_i.

Rm3 If $C_i \rightarrow C'_i$, local data need not be inaccessible (e.g., locked) during the transformation. This is because the data, even if it is updated during this process of constraint change, will, by satisfying the current constraint C_i, satisfy the eventual constraint C'_i too.

Rm4 Relocalization does not need a distributed transaction with expensive commit protocols. The two trucks had to change their local constraints in a certain sequence; but while one did so, it made no synchronization requirement on the data of the other. At each step, the global constraint was ensured.

Rm5 Suppose the sequence $< W_i >$ is broken because of disconnection. While the final local constraints would not be achieved because of the premature termination, there would not be any violation of the global constraint either (at each step of the sequence, the global constraint is satisfied).

2.3 Handling Inequality Constraints

Our example generated four inequality constraints **P1..P4**: two linear inequalities on two variables μ_1 and μ_2 and two quadratic inequalities on four variables μ_1, μ_2, σ_1^2, and σ_2^2.

Any inequality constraint C = $p(x_1, \ldots, x_N)$, where each x_i can be represented by a real number, defines a domain $Dom(C)$ in the N-dimensional space in the Cartesian coordinate system, with the i-th coordinate for x_i. The constraint C can be geometrically interpreted as: *the* datum $(x_1, \ldots, x_N)$ satisfies C *if* and *only if the* point $(x_1, \ldots, x_N)$ in *the* N-dimensional space is in $Dom(C)$. Now, suppose we are able to find $R_1, \ldots, R_N$, each a range of $\mathcal{R}$ such that

$$(x_1 \in R_1) \wedge \ldots \wedge (x_N \in R_N) \rightarrow [(x_1, \ldots, x_N) \in Dom(C)].$$

The right hand side of the above is C and the LHS is a sufficient condition SC for C; further, since each conjunct is local, we establish localization. Of course, this begs the question how these R_i can be found. Geometrically, the same LHS defines a rectangular *subset* of $Dom(C)$. All we need to do for localization therefore is to find and maintain a (N-dimensional) rectangle that is contained within $Dom(C)$ (intuitively, the closer it is to $Dom(C)$, the better). Once we find such a rectangle, the node in charge of x_i needs to maintain its data value within a range that is the projection of the rectangle on the axis x_i. Relocalization allows the change of one rectangle into another making sure that all

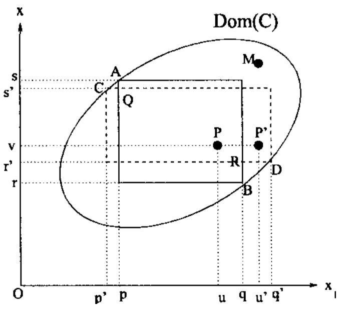

Figure 1: Quadratic Inequality

intermediate rectangles are contained in $Dom(C)$. Thus, the geometric approach reduces to rectangle management.

Let us illustrate the geometric approach with quadratic constraints. Consider a distributed constraint C of the form

$$A_1x_1^2 + A_2x_1x_2 + A_3x_2^2 + A_4x_1 + A_5x_2 + A_6 < 0 \quad \text{(or} > 0),$$

indicating a region bounded by a conic section or two parallel lines. Suppose by analysis [15] we find that $Dom(C)$ is the interior of an ellipse. We then find a rectangle which is well-oriented (i.e., has sides parallel to the x_1-x_2 axes), inside the ellipse, and is maximal (enlarging would make the rectangle extend beyond the containing ellipse). The interior of the rectangle represents the subset we are seeking. Figure 1 shows such an ellipse containing a well-oriented maximal rectangle with diagonal AB whose projections on the x_1- and x_2-axes give the local constraints $(p < x_1 < q)$ and $(r < x_2 < s)$. Computation of such a rectangle is a simple matter for conic sections.

Suppose the current global datum is $P(u, v)$. Now let a local transaction at node 1 attempt to change x_1 from u to u' which is greater than the local bound q. It is effectively attempting to move P to P', which is not in rectangle AB but still inside the ellipse. Node 2 using the value u' and its own bounds, then computes a new rectangle (shown dashed) with diagonal CD (since there is no unique solution, we use heuristics [11]), whose projections on the axes x_1, x_2 are the new local constraints. It first restricts its own bounds, which it can do unilaterally (Rm 2), thus shrinking the rectangle to QR, and then informs Node 1 that it can increase its bound and thereby enlarge the rectangle to CD.

The above example applies to P3 and P4 with one difference: P3, P4 involve 4 variables (not 2); thus, our rectangle is 4-dimensional. At any moment, the projections of that rectangle on the four axes $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2$ will give us the independent bounds on each of these four variables. While four variables are involved, μ_1 and σ_1^2 are on one machine and μ_2 and σ_2^2 are on another. There is a short-cut based on approximation that lets us revert to 2 dimensions. This is based on accepting a common bound on the variance at each node, i.e., $0 \leq \sigma_i^2/\mu_i \leq L$, for $i = 1, 2$, where L is a constant. Using this, P3 and P4 reduce to the form of C above based on two variables μ_1, μ_2 instead of four. Then the above example applies verbatim.

3. EXTENDING THE TWO-TIER MODEL

Let us outline how localization re-establishes the applicability of the two-tier model when there is a constraint relating a mobile mastered data with data on some other node (mobile or stationary). Our solution has three ingredients: **strong** local constraints, deadlines on constraint validity, and **constraint guardians.**

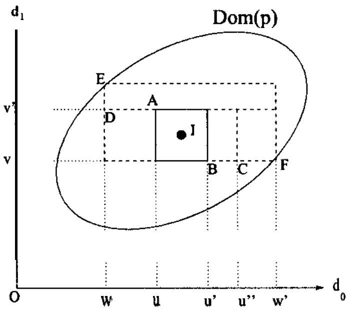

Figure 2: The Use of Non-maximal Rectangles

While in our previous discussion on rectangle management, we stated that the rectangles created should be maximal, we now make them non-maximal to get strong local constraints. The difference between this rectangle and the maximal rectangle is the leeway in the local constraint. For example, in Figure 2, AB is the initial rectangle set up given the initial value I and constraint p (in the special case where NULL values are allowed and no initial value is known, a rectangle such as AB is set up heuristically to be adjusted later dynamically based on actual values). Thus, p is localized into p_0: $u \leq d_0 \leq u'$, and p_1: $v \leq d_1 \leq v'$.

In addition, *deadlines* are set on each of these constraints. Committed tentative transactions are now guaranteed to commit when run as a base transaction provided that happens before the deadline expires, i.e., the tentativeness owing to constraint violation aborts is now eliminated. Thus, the probability of messy reconciliations is reduced.

A constraint guardian is set up to handle the local constraints, their deadlines, and their future changes. This guardian is a process that runs on the stationary network and can be invoked from any base node or from a special base node (since the base nodes are always connected, the difference between these two cases is only in performance). It exploits the leeway in case of disconnection (see below).

When a node requests an alteration (enlargement) of its local constraint, the guardian checks to see if the other node involved in the constraint is connected. If so, it guides them to negotiate a constraint change affecting both. One property of localization is that even if this negotiation is interrupted by the involuntary disconnection of one of the nodes, no inconsistency results (see Rm 5).

If the other node is not connected (it must be a mobile node), the guardian allows an alteration based on its current leeway. The deadline set is tighter if the alteration is larger. It is tightest when the alteration leads to a maximal rectangle, because there is no leeway left with the guardian. For example, after AB is set up, a change request from M_0 to accommodate a value of $D_0 > u'$ could lead to AC, and subsequently AF and DF with increasingly tight deadlines; all this without M_1 being connected. At this point, if M_1 connects and requests an enlargement, EF is the maximal rectangle that can be achieved. If this is inadequate for M_1, it will have to wait until the (tight) deadline on M_0 expires. As in the standard two-tier model, when the mobile node re-connects and submits its tentative transactions, they are run as base transactions; in addition, we require that a new local constraint be negotiated with the guardian.

Let us revisit the example in the Introduction where constraintspandq are localized: $p(D_0, D_1)$ into $p_0(D_0), p_1(D_1)$, and $q(D_2, D_3)$ into $q_2(D_2)$, $q_3(D_3)$. Because of localization, all three transactions T_0, T_1, and T_2 benefit, i.e., the two-tier model is re-established though there is a constraint relating a mobile mastered data with data on some other node. If T_0 satisfies p_0, it can commit immediately as it is a base transaction; no delays are encountered (Type A in Section 4.2). If not, then T_0 is aborted. If the abort was owing to a violation of p_0, then we check if there is a clear violation of the global constraint p (the updated value is clearly outside *Dom(p)*). If so, the transaction is rejected permanently (Type B in Section 4.2); else, the transaction is rerun later and on a violation of p_0 (it may now satisfy p_0: Type C2 in Section 4.2), an attempt is made to enlarge p_0 by contacting the guardian. If M_1 is not in contact but there is leeway, the request is granted. If the maximal rectangle has been reached or if the granted enlargement is inadequate, we have to wait until the deadline on p_1 at M_1 expires. When M_1 is in contact, but still the maximum enlargement is inadequate, then it means that the global constraint would be violated, hence the transaction should be aborted.

The same is the case with the commit for T_2. If T_2 aborts (owing to a violation of sufficient condition only), then M_2 asks for an enlargement of its constraint. The guardian will use its leeway to grant this request if M_3 is not connected. This parallels the T_0 case.

If the tentative transaction T_1 commits at M_1, it is guaranteed to commit as a base transaction provided that M_1 connects to the stationary network before the deadline expires. However, if it aborts owing to a violation of sufficient condition only, then it is enough to re-run it as a base transaction and let the guardian be invoked to negotiate an enlarged constraint if it aborts there.

4. HANDLING RELOCALIZATION

We have already discussed how the local sufficient conditions can be adjusted dynamically based on demand. The actual mechanism is complicated owing to two factors: first, the change of constraints must itself avoid a distributed transaction, and second, simultaneous requests for constraint change from more than one node can result in inconsistency. In this section, we will discuss the algorithm needed to support such constraint transformations. It can be used in a peer-to-peer mode in the stationary part of the database. In the two-tier model, it is used by the guardian when one of the nodes is a base node and the other is a mobile node, or on behalf of a mobile node which requested a constraint change and then got disconnected. When the guardian finds that both/all nodes involved in a constraint are connected, it may instruct them to negotiate directly with each other provided that they inform the guardian of the outcome. While describing the algorithms below, to keep them general, we will not mention the special role of the guardian. We present some results we obtained by implementing this algorithm and experimenting on the simultaneity problem partly on a simulated network and partly on a real network.

Again, we assume that the constraint involves $N \geq 2$ nodes. We first deal with the case without simultaneity and then discuss the complexity introduced by simultaneity. The framework for simultaneity resolution presented in this section is a significant improvement over that in [12]: (1) we present a general framework for an arbitrary number of database nodes, and (2) we avoid m-sending requests after an asymmetric delay in case of simultaneity, lowering substantially the transaction response time and use of resources.

Relocalization can be explained using procedures $Request_i$, *Consider-i*, $Accept_i$, and $ResolveSimulteneity_i$ at node i. Below, we discuss the main steps in these procedures. Pseudocodes are given in [10].

Assume that data item x_i is at node i ($1 \leq i \leq N$). Through localization, node i maintains its local condition C_i. Suppose a transaction T_i at i attempts to update x_i to x'_i. If x'_i violates C_i, T_i is blocked (or aborted). Node i enqueues a request (req_i) in a queue Q_i and asks for a change of constraints through $Request_i$, which finds req_i in Q_i and broadcasts a message to all other nodes passing req_i. This message invokes $Consider_j$ at nodes $j \neq$ i.

In the general case, req_i does not contain any data — it is just a signal that node i needs a more relaxed local constraint. $Consider_j$ (at all other nodes j) restricts (e.g., locks) x_j and heuristically computes how C_j can be "sacrificed" in order to "help" C_i. If this is possible, it updates C_j to a "shrunk" C'_j. It then frees x_j and sends a reply $rep_j = (C_j)$ (whether C_j is updated or not). Replies are received by $Accept_i$. The last reply triggers an algorithm, which attempts to compute a more relaxed C'_i given current C_j's (revised or not) at all other nodes. If successful, it upgrades C_i to C'_i and arranges for the resume (or restart) of T_i. If unsuccessful (possibly because other nodes could not "sacrifice" their constraints), $Accept_i$ just dequeues Q_i.

It is worth noting that when N = 2, the algorithm can be simplified. We can have $req_1 = (C_1, x'_1)$. Then, the only recipient of req_1 (i.e., $Consider_2$) has all necessary information to compute both C'_1 and C'_2, such that x'_1 is accommodated. Subsequently, we can have $rep_2 = (C'_1)$ if $Consider_2$'s computation was successful, and rep_2 = ("no") otherwise.

4.1 Simultaneous Requests

Undetected simultaneous initiations of constraint change could lead to inconsistency; e.g., for N = 2, Consider1 and $Consider_2$ may localize differently, yielding different SC''s: the two nodes could adopt local sufficient conditions whose conjunction does not imply the global constraint.

We will say that node i is involved in an occurrence of simultaneity when at least one request req_j is received by Consider;, while there is a request req_i sent out from node i for which no reply has been yet received. In such cases we will say that req_j is simultaneous *with* req_i. Clearly, while waiting for replies for req_i, node i may receive the awaited replies from some nodes, and requests simultaneous with req_i from others. We will refer to the set of nodes that sent requests, plus the node i itself, as the *simultaneity cluster.*

Our solution to the problem of simultaneous requests is to ensure that there is at most one simultaneity cluster at a time and all nodes involved in an occurrence observe the cluster correctly. Having achieved this, we can force sequential processing of the requests among nodes in the cluster.

Simultaneity can be detected by setting a flag whenever there is a request sent out from node i for which no reply has been received and checking it when a request is received from another node. Such detection is symmetric (if message order is preserved) with respect to each pair of nodes.

However, this alone will not solve the problem. For ex-

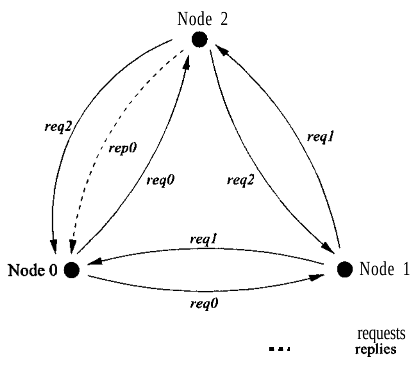

Figure 3: Simultaneity detection

ample, let N = 3 and consider the scenario described by the following sequence of events and depicted in Figure 3: Node 0 broadcasts a request (req_0). Node 1 follows with (req_1). Node 2 receives req_0 and sends a reply (rep_0). It then broadcasts a request (req_2). Node 0 receives rep_0 and req_1 and detects simultaneity with node 1. Next, node 2 receives req_1 and detects simultaneity with node 1. Finally, node 1 receives req_0 and req_2 and detects simultaneity with nodes 0 and 2. At this point, we have inconsistent simultaneity detection: nodes 0 and 2 detected simultaneity with node 1 only, while node 1 detected simultaneity with both 0 and 2. Since the observed simultaneity clusters are different at each node, sequential processing of requests is impossible.

To cope with this problem, we introduce mandatory acknowledgements for replies to non-simultaneous (or regular) requests. A node replying for a regular request cannot send its own request until an acknowledgement is received from the requester. The acknowledgement will be sent if the requester it is not involved in simultaneity, or simultaneity has been resolved. Thus, in the above scenario, req_2 cannot be broadcast because node 2 now must wait for an acknowledgement for rep_0 from node 0. Consequently, node 2 would receive req_1 and send a reply to node 1 (which means that node 2 must now wait for 2 acknowledgements). Node 1, in turn, would receive that reply and req_0, and detect simultaneity with node 0. The acknowledgements from nodes 0 and 1 would be sent only after both nodes resolve the correctly observed simultaneity.

It can be proven that, when the above acknowledgement regime is enforced, involvement in simultaneity is both symmetric and transitive, and that there can be at most one simultaneity cluster at a time, i.e., our problem is solved. Note that the above acknowledgement scheme can be ignored when N = 2, since here the simultaneity cluster always has 2 nodes in it, i.e., the above problem is nonexistent.

To handle simultaneity, $Consider_i$ and $Accept_i$ recognize two phases: a detection phase and a resolution phase. The detection phase at node i starts when $Consider_i$ receives the first request simultaneous with its own request and ends when the node finishes computing the simultaneity cluster. During this phase, the node waits for messages from all other nodes: a reply (received by $Accept_i$) indicates that the sender is outside of the cluster, and a request (received by $Consider_i$) indicates that the sender is inside the cluster. The resolution phase starts when the node has received messages from all other nodes. A queue SQ of nodes from the cluster (identical at all nodes) is created; SQ is a sub-list of NL (the list of all nodes). Then, simultaneous requests are processed in a sequence induced by SQ.

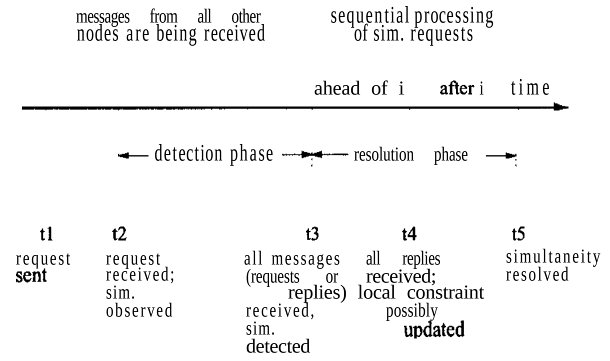
messages from all other nodes are being received
sequential processing of sim. requests
ahead of i
after i
time
detection phase
resolution phase
t1
request sent
t2
request received; sim. observed
t3
all messages (requests or replies) received, sim. detected
t4
all replies received; local constraint possibly updated
t5
simultaneity resolved

Figure 4: Simultaneity resolution at node i

Let us consider a typical scenario of simultaneity resolution at node i depicted in Figure 4. At time t_1, a request r_i is sent by node i. From now on, node i waits for messages from all other nodes. At t_2, the first request (r_j) simultaneous with r_i is received. Thus, at least two nodes (i.e., j and i itself) are in the simultaneity cluster. At t_3, messages from all nodes are in, completing the detection phase.

Now, an asynchronous process ($ResolveSimulteneity_i$) conducts the resolution phase. Let $k = Head(SQ)$. If $k \neq i$, $ResolveSimulteneity_i$ processes the request from node IF, sends a reply, and dequeues SQ. All other involved nodes except k do the same so that k gets replies from all. If $k = i$, it is now node i's turn in the sequential processing. $ResolveSimulteneity_i$ blocks until all replies are in (t_4 in Figure 4). It then possibly updates C_i, resumes or restarts the originating user transaction, sends all required acknowledgements, and dequeues SQ. This loop ends when SQ is empty (t_5 in Figure 4). The simultaneity incidence is resolved.

To avoid having some nodes suffer from being at the end of SQ every time, NL is rotated circularly after each simultaneity occurrence. Nodes outside of the cluster are notified of the new value of NL by piggybacking it on the acknowledgements (that those nodes must wait for).

Interestingly, the acknowledgements do not degrade the performance (in terms of transaction response time) as much as one may suspect. In fact, when relocalization requests are not frequent, it is the replying node that has to wait for acknowledgements, but this does not hold up any of its transactions. On the other hand, when relocalization requests are frequent, simultaneity is likely, but acknowledgements are required for replies to regular requests only.

4.2 Categorizing User Transactions

We categorize all user updates at node i as follows:

type A — Update satisfies C_i.

type B Update cannot satisfy any C_i.

type C — Update violates C_i and req_i is enqueued in Q. Subdivided into:

- type **C1** — req_i is processed avoiding simultaneity.
- type **C1s** — req_i results in a simultaneity occurrence.
- type **C2** — Owing to a beneficial relocalization (due to a request ahead of req in Q), the update now satisfies C_i.

Types A, B, and C2 are processed locally; Cl and Cls require 1 network "round-trip" message (request and reply). An acknowledgement is required for type Cl when N > 2. For N = 2, we ignore acknowledgements, and subdivide type Cls into Clsc when $SQ = (i, j)$, and Clsw when SQ = (j, i).

4.3 The Experiment

We implemented the above procedures as asynchronous processes on Pentium-class machines running Linux. We simulated a transaction environment (using additional procedures) while sharing a constrained data element, performing relocalization as needed, allowing us to monitor the effects of simultaneous requests. Our implementation processed messages in the order they were sent (for N > 2, this was a pair-wise condition). For N = 2, we simulated the network delays on a single machine and used the simplified version of relocalization (described in Section 4); for N > 2, we used a real network of similar machines.

We first report on an experiment in which N = 2, C = $x_1^2 + x_2^2 < 4$, and initially $(x_1, x_2) = (0, 0)$. Next, we present our studies on the effect of simultaneity clusters for N > 2 without reference to any particular constraint.

4.3.1 Results for N = 2

We observed the outcomes of various user updates, and measured actual times on the simulated network. We relied on the operating system to handle buffering. This closely approximated the relative timings involved in a typical database operation, i.e., delays due to context switches, locks, sending/receiving messages, etc. are taken care of. However, since delays from/to actual user transaction were missing, we added a compensating adjustment of 0.2 ms. Two parameters were chosen to explore the algorithm's performance

- user *speed* is an interval *(minsleep,maxsleep)*. The time between successive update transactions is a random value within this interval.
- user restraint r is the degree of restraint in update: $x' = x + w.g/r$ relates the updated data x' with the current value x, where w is a random value ($-1 \leq w \leq 1$), and g is a constant.

Three user speeds were chosen: (50, 1000), (10, 20), and (0.1, 5), all in ms, and denoted *User1*, *User2*, and User3 respectively; g was 4 (the diameter of the constraint circle), and r varied from 0.5 to 5.0 in increments of 0.5, and then to 10.0 in increments of 1. For each user speed (Users 1, 2, and 3) and every r, 15 minute-long simulations were run.

In Figure 5, for User 3 and each r, we see a bar showing the percentages of transactions in each of the categories (types A, B, Cl, Clsc, Clsw, and C2). A large majority of transactions (close to 75%) do not need any requests because of localization. For large r, most user updates are executed locally; for small r, they are rejected locally. For a certain value of user restraint close to 2, there is a maximum of user updates needing request messages to be sent over the network (types Cl, Clsc, and Clsw).

In Figure 6, we plot the average response time (the time from the moment a request has been enqueued to the moment it either has been re-executed successfully after relocalization or is eliminated after a "no" reply) for type C requests against r, for each user speed. We observe that a slow user has a reasonably constant response time of roughly the network delay, which, in our experiment, was 20 ms each

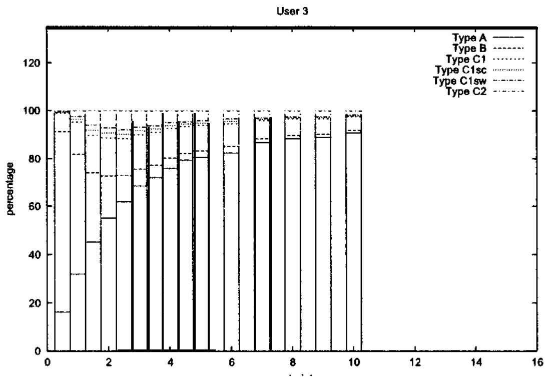

Figure 5: Distribution of user updates; User 3

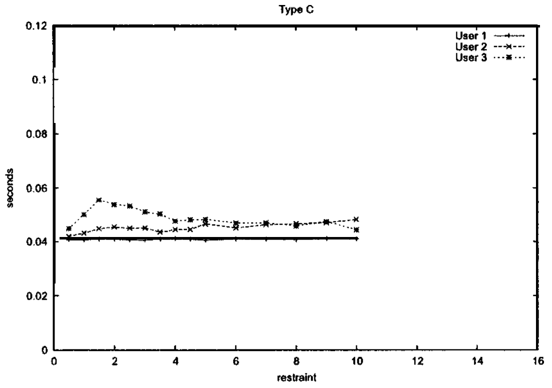

Figure 6: Average response time for Type C

way. This is because when a constraint change request is sent, it is mostly resolved without simultaneity (type Cl). For a fast user, the response time is higher due to simultaneity and the need to delay buffered requests.

In Figure 7, we plot the average response time for all requests. Comparing with Figure 6, one sees that the overall response times are much smaller. The reason is that a majority of user updates are processed locally.

Based on our observations and analysis, the only category with a significantly higher response time than just the network delay is Clsw those requests that wait in case of simultaneity. But, this category makes up only half of the simultaneous requests. The other half (type Clsc) have approximately equal response time to that of regular requests. To summarize, localization is clearly useful; simultaneity is important but our approach at tackling it is effective.

4.3.2 Results for N = 5

Using five machines on a network, we simulated Users 1, 2, 3, and 4 using sleep intervals (1 s, 2 s), (0.5 s, 1 s), (0.1 s, 0.5 s), and (50 ms, 0.1 s) respectively. Here each transaction had a 25% chance of violating the local constraint. When requests were considered, no constraint changes were computed since we were only interested in the simultaneity issue. Note that due to the absence of the actual constraint, in this experiment category C2 is missing.

In Table 1, we tabulate for each user U, *reqs* (the number of user updates during the whole simulation), C (the % of

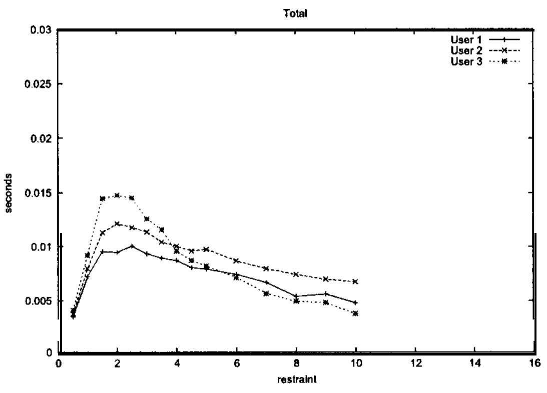

Figure 7: Overall average response time

type C updates), sent (the % of type C updates that resulted in a constraint update request; C2 is missing, but if the user is fast Q is non-empty when the simulation stopped), sim (% of requests sent that were simultaneous), and sim n (% of simultaneous requests in cluster size n for n = 2..5).

In Table 2, we report the response times of all type C requests that were sent (C), of non-simultaneous requests (*nonsim*), of simultaneous requests for all cluster sizes (sim), of simultaneous requests (cluster size n) (*sim n*). All times are averages in seconds.

These two tables show that cluster size increases with the user transaction rate. However, smaller clusters are much more frequent than larger ones and with our algorithm, the time degradation due to simultaneity is not significant.

U	*reqs*	C	sent	sim	*sim2*	*sim3*	*sim4*	*sim5*
1	191	23	100	7.0	100	0	0	0
2	379	25	100	11.6	100	0	0	0
3	944	25	100	18.5	97.7	2.3	0	0
4	3270	27	99.8	60.1	67.5	26.5	5.3	0.8

Table 1: Distribution of Simultaneity

U	*C*	nonsim	sim	*sim2*	*sim3*	*sim4*	*sim5*
1	0.07	0.08	0.07	0.07	0	0	0
2	0.06	0.06	0.13	0.13	0	0	0
3	0.06	0.06	0.07	0.07	0.13	0	0
4	0.07	0.05	0.08	0.07	0.10	0.14	0.10

Table 2: Response Times

5. CONCLUSION

For data replication in a mobile environment, the two-tier model had been shown to be the best compromise maintaining strict consistency while minimizing the effects of reconciliation by limiting the kinds of transactions. The contributions of this paper are twofold.

1. Instead of relying on ad-hoc methods for transactions outside the scope of the two-tier model, our approach based on localization extends the applicability of the model to transactions as typified by T_0, T_1, and T_2 in a systematic manner. Localization involves reformulation of constraints into local sufficient conditions thereby enhancing the node autonomy;
2. We introduce the notion of a constraint guardian, which we view as a proxy for metadata management as opposed to the traditional proxy which is used for the management of data on behalf of a mobile node and is less lean and flexible. Also, we have implemented and tested algorithms for relocalization taking care of the problem of simultaneous constraint change; we present those results.

6. REFERENCES

[1] R. Alonso, D. Barbara, and H. Garcia-Molina. Data Caching Issues in an Information Retrieval System. *ACM TODS*, 15(3):359–384, Sep. 1990.

[2] D. Barbará and H. Garcia-Molina. The Demarcation Protocol: A Technique for Maintaining Linear Arithmetic Constraints in Distributed Database Systems. In *Proc. EDBT Conf.* pp. 373-388, 1992.

[3] M. Franklin, M. Carey, and M. Livny. Transactional Client-Server Cache Consistency: Alternatives and Performance. *ACM TODS*, 22(3):315–363, Sep. 1997.

[4] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Proc. SIGMOD Conf.*, pp. 173-182, 1996.

[5] N. Krishnakumar and A. Bernstein. High Throughput Escrow Algorithms for Replicated Databases. In *Proc. 18th VLDB Conf.*, pp. 175-186, 1992.

[6] A. Kumar and M. Stonebraker. Semantics-based Transaction Management Techniques for Replicated Data. In *Proc. SIGMOD Conj.*, pp. 117-125, 1988.

[7] J. Lloyd. Foundations of *Logic* Programming. Springer-Verlag, 1984.

[8] S. Mazumdar. Optimizing Distributed Integrity Constraints. In *Proc.* 3^{rd} *Intl.* Symp. on Database Systems for Advanced Applications, pp. 327-334, 1993.

[9] S. Mazumdar and P. Chrysanthis. Achieving Consistency in Mobile Databases through Localization in PRO-MOTION. In *Proc. DEXA-MDDS Wkshp.*, pp. 82-89, 1999.

[10] S. Mazumdar, M. Pietrzyk, and P. Chrysanthis. Caching Constrained Mobile Data. TR-CS 01/5/401, New Mexico Tech, May 2001.

[11] S. Mazumdar and G. Yuan. Localizing a Class of Distributed Constraints: A Geometric Approach. *J.* of Computing and Information, 3/ICCI98/6/2, 1998.

[12] M. Pietrzyk, S. Mazumdar, and R. Cline. Dynamic Adjustment of Localized Constraints. In *Proc. DEXA Conf.*, pp. 791-801. 1999.

[13] E. Pitoura and G. Samaras. *Data* Management for *Mobile* Computing. Kluwer, 1998.

[14] K. Bamamritham and P. Chrysanthis. Advances *in* Concurrency Control *and Transaction* Processing. IEEE Computer Society Press, 1996.

[15] L. L. Smail. Analytic *Geometry* and Calculus. Appleton-Century-Crofts Inc., 1953.

[16] N. Soparker and A. Silberschatz. Data-value Partitioning and Virtual Messages. In *Proc. PODS* Symp., pp. 357-367, 1990.

[17] G. D. Walborn and P. Chrysanthis. PRO-MOTION: Support for Mobile Database Access. *J.* of Personal Technologies, 1(3):171–181, 1997.