Mobile Networks and Applications 9, 481–490, 2004

Localization of Integrity Constraints in Mobile Databases and Specification in PRO-MOTION

SUBHASISH MAZUMDAR
Computer Science Department, New Mexico Tech, Socorro, NM 87801, USA

PANOS K. CHRYSANTHIS
Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260, USA

Abstract. The well-developed traditional data management techniques need to be augmented with new approaches in order to continue to be effective in the mobile environment. In this paper, we focus on the challenge of maintaining integrity constraints in the presence of disconnections and expensive communication. Our approach of localization is to reformulate global constraints so as to enhance the autonomy of the mobile hosts in processing transactions. We show how this approach unifies techniques of maintaining replicated data with methods of enforcing polynomial inequalities. We also discuss how localization can be realized in PRO-MOTION, a flexible infrastructure for transaction processing in a mobile environment.

Keywords: mobile databases, disconnected operations, transaction processing, data consistency, integrity constraints, caching and replication

1. Introduction

Thanks to the relentless advances in semiconductors, the number of users with mobile computers (we will refer to these machines as *mobile hosts* or MHs) continues to increase. These users have discovered that exciting developments in wireless technology can potentially empower them to access remote information *anywhere, anytime*, and *in any way*. To be truly effective, however, users of MHs need the ability to both query and update public as well as private corporate databases. These databases typically execute atomic transactions to assure data consistency and reliability in spite of concurrent updates and system failures. Thus, transaction support must be extended to mobile users [19].

Owing to portability requirements, mobile computers are, in general, less robust than stationary ones. Not only are they prone to physical hazards, but also suffer from limited battery life, reduced storage capacity, and a relatively low-bandwidth, expensive, and tenuous wireless connection to the fixed network. They may become disconnected when placed in certain terrain or enclosure. Also, a MH may choose to power down *only* the communication subsystem to save battery life or communication dollars; while such a MH is disconnected, it has *not failed* for it can continue processing [2,8]. Such transience in connectivity, we argue next, makes transaction processing a challenging task.

The basic problem is that a mobile computer must share some data item D with a database in the fixed network, and consequently agree to satisfy an integrity constraint C that ensures correctness of shared components. C is distributed (or global) since it spans at least the mobile computer and one other database. Consider a local transaction T executing on the mobile host; if it accesses D, it must, when it tries to commit, verify that C is preserved, i.e., that it holds at the end of the transaction. But this verification implies a query which involves at least one other host – so, T is no longer local: it is *distributed!* Consequently, it invites the expenses and problems associated with distributed transactions: network communication, distributed concurrency control for synchronization of remote data, and commit protocols [3]. While complex, this is a minor matter because all this can be accommodated by a Distributed DBMS. The major problem is that in the mobile environment, there is an additional factor, the whimsical connectivity of mobile hosts, owing to which, *unbounded and unpredictable delays* can afflict not only T but *other transactions* running at *both* the mobile and the stationary node(s). This is clearly unacceptable. Extending transaction management and data consistency maintenance to cover disconnected and mobile operations is the challenge we address.

Our approach is pre-emptive: when the MH above shared D, it agreed to a *global* constraint C; our aim is to give it a *local* constraint C' instead. For the MH, C' could very well be more restrictive than the original C, but this is the tradeoff against the enhanced autonomy brought by the locality of C'. As a reflection of this autonomy, the local transaction T would remain local; thus, the unacceptable delays discussed above would be avoided.

By a process we call *localization*, we reformulate a distributed constraint into local constraints and adjust them dynamically. We have looked at two kinds of constraints – based on equality and inequality respectively (for set-based constraints, see [12]). Localization provides a framework for a number of well-known but disparate techniques of concurrency control such as *lease*, *callback*, and *check-in/check-out* [5,20] for equality constraints on replicated data, the *Escrow* method

[9,10,17] and the *Demarcation Protocol* [1,4] for linear inequality constraints; it has also enabled a hitherto unknown extension of Escrow/Demarcation Protocol to quadratic inequalities.

The conceptual framework of localization blends synergistically with an architecture similar to that of PRO-MOTION, a flexible infrastructure for transaction processing in a multitier, mobile client–server operating environment [13,22]. By caching data items locally, it allows MHs to continue executing transactions while disconnected from the stationary server; it incorporates the modified data back into the stationary server's database when reconnection occurs. The cached data is in the form of an encapsulated object containing data bits, methods, rules, and state information. This object is called the *compact*. Localization allows us to fill in certain components of appropriate compacts enabling unilateral commitment of local transactions.

In the next section, we present a running example. Then we introduce PRO-MOTION; next, we explain localization. In the following two sections, we apply the technique to dynamic replication and polynomial inequalities respectively. Finally, we outline how the localized constraints would be handled in PRO-MOTION.

2. An example

Mobile computers are becoming more and more common in the trucking industry [23]. Each truck has a computer with a satellite or radio link. It not only communicates with a corporate database, but is also used for billing and gathering data from various vehicle instruments; it may even be used to transmit funds directly for the driver's expenses.

Consider a trucking company that has accepted a contract to move manufactured goods from a source to a destination. It, in turn, subcontracts privately owned trucks by posting a list of contracts to which truck operators respond with their offers; typically, one such offer is selected for each contract, though in some cases, the company may prefer to fulfill a contract through subcontracts based on two or more offers. The driver of a selected truck then receives a shipping manifest, goes to the source to pick up the material, and later to the destination to deliver them. The shipping manifest specifies the quantity of parts to be picked up plus pertinent information about the truck, the driver, and the source.

We envision a pick-up process during which goods are also checked to see if they meet their specifications, i.e., if an attribute A (e.g., diameter of a washer) is within its specified tolerance. The motivation is to perform quality control during pick-up itself so as to avoid returning unsatisfactory goods later because the process of return is costly in terms of both time and money. Thus, the truck's mobile computer measures the mean and variance of A. If these two metrics are outside their acceptable range, the goods are rejected on the spot. Assuming that two trucks are awarded partial contracts, we have an added complication: the destination will merge the two truckloads and the *overall* mean μ and variance σ^2 of *A of the merged collection* must be within tolerable limits: $M_0 \leqslant \mu \leqslant M_1$ and $0 \leqslant \sigma^2/\mu^2 \leqslant K$, where M_0, M_1, K are constants (we capitalize constants).

When the load is delivered, the driver records the date and time, obtains a signature from the receiving party for billing. We label the important steps OFFER, MANIFEST, PICKUP, and BILLING, respectively. Their ramifications on transaction support and consistency maintenance are as follows.

OFFER. The list of contracts is posted on the company's database and refreshed periodically. Any truck can access this list for review. If an operator wants to offer its service, it requests and obtains exclusive write privilege, betters whatever offers are currently available and hands it over for the company and other operators to review (anonymity of operators is ensured if necessary). Exclusive write privilege is needed since an operator who stops on the road to make an offer does not want to see it concurrently overwritten by another.

Suppose the original contract was for a load *Qty*. Trucks 1 and 2 (perhaps among others) responded with offers for loads *Qty1*, *Qty2* (as much as their trucks could carry), but they were awarded partial contracts for Q_1, Q_2, respectively, where $Qty = Q_1 + Q_2$ and $Q_1 \leqslant Qty_1$ and $Q_2 \leqslant Qty_2$.

Clearly, it is not in the interest of an operator who wants to make offers to remain disconnected for long. On the other hand, indefinite disconnection by a writer can hold up other operators.

PICKUP. 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 the constraints in table 1 (two linear and two quadratic polynomial inequalities in four variables $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2$).

So, our truck driver after measuring μ_1, σ_1^2, will attempt to verify P1–P4. But to do so, he/she must access μ_2, σ_2^2 measured by the second truck. Owing to disconnection, even if that second truck performs its measurements at the same time, this verification may incur unpredictable delays.

MANIFEST. The manifest information should be made permanent in the company database before the truck is permitted to travel. Next, it should be replicated in the mobile computer to be looked up when needed. The quantities of material to be picked up Q_1, Q_2 are replicated on the

Table 1
PICKUP constraints.

P1	$R_1\mu_1 + R_2\mu_2 \geqslant M_0$
P2	$R_1\mu_1 + R_2\mu_2 \leqslant M_1$
P3	$R_1\sigma_1^2 + R_2\sigma_2^2 + R_1R_2\mu_1^2 + R_1R_2\mu_2^2 - 2R_1R_2\mu_1\mu_2 \geqslant 0$
P4	$R_1\sigma_1^2 + R_2\sigma_2^2 + R_1(1 - R_1 - KR_1)\mu_1^2 + R_2(R_1 - KR_2)\mu_2^2 - 2R_1R_2(1 + K)\mu_1\mu_2 \leqslant 0$

databases of the first and the second truck, respectively. Disconnection is of little consequence here.

In some cases, e.g., if more goods are available and the truck has space, its operator may request a change in the quantity to be picked up. Processing this request is non-trivial since it involves not only extra (or less) goods but also change in the constraints themselves (Q_1 and Q_2 both affect R_1, R_2, which occur in P1–P4) affecting both trucks. Clearly, disconnection during such a change in quantity can hold up the other truck's pickup process.

BILLING. The delivery information can be finalized at the mobile computer and incorporated in the company database shortly thereafter. A disconnection is not catastrophic: it will only postpone the billing process.

3. PRO-MOTION

PRO-MOTION is ideally suited to realize applications such as the one above using our approach of localization. It provides a flexible infrastructure for transaction processing in a multi-tier, mobile client–server environment [22].[1]

We assume a general mobile computing environment in which the network consists of stationary and mobile hosts (MHs) [7]. Certain specialized stationary hosts called *Base* or *Mobility Support Stations* (BSS/MSSs) are equipped with wireless communications capabilities that enable the mobile hosts to connect to them, and through them, to the high-speed fixed network. At any moment, a MH is either connected to the network through a specific MSS based on its location or completely disconnected.

The goal is to process as much of a transaction as possible on the MH, resorting to communication with the stationary database server (SDS) only when convenient or when absolutely required by the semantics of the transaction. This is achieved in PRO-MOTION by replicating or caching data from the server; such replicated data is always in the form of an encapsulated object called a *compact* (figure 2). A compact is, broadly speaking, a satisfied request to cache data, enhanced with *obligations* (e.g., a deadline), *methods* (a set of allowable operations), *state information* (e.g., the time of last update), and *consistency rules* (restrictions on possible states). Unlike mobile agents, compacts are active objects

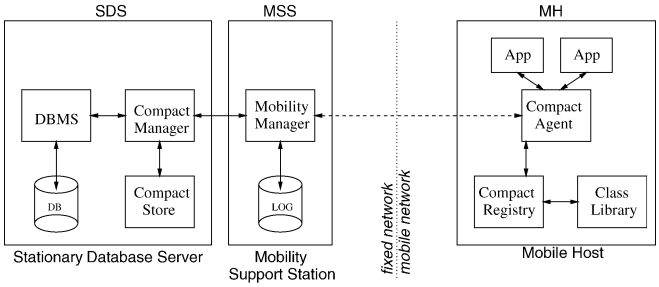

Figure 1. PRO-MOTION system architecture.

[1] The simplest form of multi-tier mobile client–server architecture in PRO-MOTION is currently referred to as the client–intercept–server model [6].

that are invoked in the context of a transaction and controlled by the Transaction Manager. Compacts are supported at the SDS, the MSS, and the MH as follows.

- At the SDS, there is a *compact manager*. It acts as a front-end, shielding the server from the idiosyncrasies of the mobile environment.
- At the MSS, there is a *mobility manager*, which helps manage the communication flow between the compact manager on the server and the compact agent (see below) on the MH. A MH can send an update message and disconnect immediately relying on the mobility manager to pursue the update on its behalf and store the acknowledgement.
- At the MH, there is a *compact agent*. It negotiates with the mobility manager, manages compacts, and acts as a transaction manager for transactions executing on the MH. It also handles disconnections and manages storage.

The compact represents an agreement between the server and the MH in which the server delegates control of the data to the MH which pledges to honor specific conditions regarding its use as set by the server. Compacts are obtained from the server via *requests* from the MH (to fill an imminent or anticipated data need). If the request can be satisfied, the server's compact manager creates a compact containing data plus information required for its correct usage. The compact is recorded in a *compact store* and transmitted to the MH. The request from the MH can be tailored to cause only incremental transmission. For example, transmitting the compact methods, which may be very expensive, is avoided if they are already available on the MH. Once the MH receives the compact, it records it in a *compact registry* which is used by the compact agent to track the location and status of all active compacts.

When the needs of the mobile host or the database server change, compacts may be *renegotiated* to redistribute resources; when the MH no longer needs the resources, compacts are *returned* to the database server and deleted from the local compact registry and the compact store.

Each compact has a common interface which is used by the compact agent to manage the compacts listed in the compact registry and to perform updates submitted by transactions run by applications executing on the MH. The basic set of methods necessary to manage compacts are the following:

- inquire() retrieves useful information about the compact state, e.g., name, data type, version, cache status, free storage, and outstanding transaction IDs;

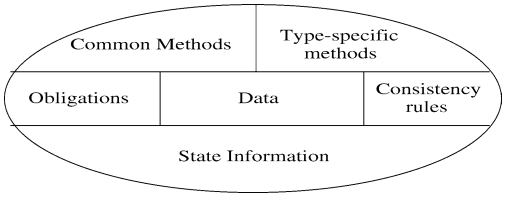

Figure 2. Compacts as objects.

- dispatch() performs operations on the compact on behalf of transactions executing on the MH;
- checkpoint() stores the current state of the compact for recovery purposes;
- commit() makes the operations of a transaction permanent on the compact (local commit), and ultimately on the database server (global commit);
- rollback() abandons a transaction's modification of the compact data; and
- notify() tells the compact that the mobile environment has changed, e.g., when a local transaction must be re-done or a parameter re-negotiated.

4. Localization

Let us now elaborate on the notion of localization. We assume that data is distributed among nodes $1, 2, \ldots, N$. In section 7, we will relate these nodes to mobile hosts and the stationary server. 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 $(1 \leqslant i \leqslant N)$ is said to be *localizable* if there is a rule

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

such that $C_i(\overline{x_i})$ is local for $1 \leqslant i \leqslant N$. Substitution of the distributed constraint C by local constraints C_i at each node i $(1 \leqslant i \leqslant N)$ is referred to as *localization*.

Variables and quantifiers (not shown) conform to the rules for Horn clauses [11]. We ignore trivial rules where one or more of the C_is are false.

We denote the left-hand side 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 $1 \leqslant i \leqslant N$.

For example, let C be P1: $R_1\mu_1 + R_2\mu_2 \geqslant M_0$ (table 1). It is a distributed constraint with $N = 2$ (the two nodes are the two trucks) on variables μ_1 and μ_2. Using the rule $C_1 \wedge C_2 \rightarrow P1$, where $C_1 = (\mu_1 \geqslant L_1)$, $C_2 = (\mu_2 \geqslant L_2)$, and L_1, L_2 are constants such that $R_1L_1 + R_2L_2 \geqslant M_0$, we see that P1 is localizable. Thus, we can enforce $\mu_1 \geqslant L_1$, a local constraint, at the first truck, and $\mu_2 \geqslant L_2$ at the second, 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 transformable to, say, SC' that could accommodate the updated value. SC' would be of the form:

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

Though the transformation of SC to SC' typically involves constraint changes at more than one node, we want this to be achieved in an incremental node-by-node manner, perhaps in some pre-determined order, so that synchronization delays due to commit protocols arising from distributed transactions can be avoided.

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 $0 \leqslant i \leqslant n$ 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

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

This is basically what is achieved by the Escrow method and Demarcation Protocol but these methods do not work for constraints P3 and P4.

Note that 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 first truck decreased L_1 before the second truck increased L_2), the resulting intermediate state with W_1 replaced by $(\mu_1 \geqslant L'_1) \wedge (\mu_2 \geqslant L_2)$ may not have satisfied P1.

In other words, localization allows a distributed constraint to be replaced by a bunch of local constraints and relocalization allows their dynamic adjustment while always implying the original constraint. The following remarks cover useful properties and indicate why node autonomy is enhanced.

Remark 1. If a local transaction at node i satisfies C_i, no global constraint needs to be checked either during execution or commit and therefore unpredictable delays are avoided.

Remark 2. In general, a node i cannot unilaterally change its local constraint C_i. However, if $C'_i \rightarrow C_i$ (the new constraint C'_i is more restrictive), it can change unilaterally provided the data which now satisfies C_i does also satisfy C'_i. For example, the second truck could increase L_2 to L'_2 unilaterally.

Remark 3. If $C_i \rightarrow C'_i$, local data need not be locked[2] during the transformation from C_i to C'_i because the data, even if it is updated during this process of constraint change, will, by satisfying the current C_i, satisfy the new C'_i too.

[2] We are not presuming that locking is the concurrency control mechanism. The remark is applicable no matter what kind of synchronization mechanism is used.

Remark 4. In the special case where $C_i(x)$ means that x *is undefined* and C_i' means that x *is defined and satisfies predicate* p, the transformation from C_i to C_i' consists of assigning a value to x that satisfies $p(x)$; no local locking is involved. Similarly, if the transformation goes the other way, the current value of x needs to be invalidated; no locking is involved.

Remark 5. Relocalization of SC can be done one node at a time, possibly in a certain prescribed sequence, but no distributed transaction with expensive commit protocols is needed. In the example, the two trucks had to change their local constraints in a certain sequence; but while a node changed its constraint, it made no synchronization requirement on the data of the other node.

Remark 6. Suppose the sequence $\langle W_i \rangle$ is broken because of a unilateral decision made by a node to terminate the process of relocalization. While the final local constraints would not be achieved because of the premature termination, there would be no violation of the global constraint because at each step of the sequence, the global constraint is satisfied.

Remark 7. When two or more nodes initiate relocalization concurrently, the maintenance of correctness while avoiding distributed transactions is not simple – we discuss our algorithms and their performance elsewhere [14,18].

Remark 8. We have shown recently [14] that localization can be used to extend the scope of the popular two-tier model in which mobile devices form one tier and always-connected stationary servers the other.

5. Localization and dynamic replication

In this section, we first review the caching idea from the localization perspective and see that invalidation is *not* necessarily a requirement of modification. Data replication semantics such as *callback, lease, check-in/check-out* can all be captured under the framework of localization and relocalization. We dwell only on lease semantics illustrating a temporal dimension of localization in the process.

5.1. The caching approach

Equality constraints are generally of the form $x \equiv f(y)$ where x, y are data items and f is some function (for pure replication, f is the identity function and $\equiv$ is equality). When x and y are at different nodes (sites), this is a distributed constraint that is typically handled in caching schemes by following the rule: either allow x and y to be read-only or invalidate one of them so that the other can be modified. Of course, this scheme can be modified to apply to more than two variables. Further, although we assume that all communication is peer-to-peer, it is also applicable in settings in which communication is facilitated by a server. As we will see in section 7, such communication with a facilitator server is more appropriate for mobile databases.

Suppose a logical item X has replicas x_i at node i (for various i). Not all the replicas are defined at all times; we denote the predicate *data item A is defined (undefined)* by $A\uparrow$ (respectively $A\downarrow$). The condition C for integrity of replicas can be stated as

$$C: \quad (\exists j)(\exists v)\big[(1 \leqslant j \leqslant N) \wedge x_j\uparrow\big] \bigwedge_{i=1}^{N} \big[x_i\uparrow \rightarrow (x_i = v)\big].$$

This condition C has two parts; the first part states that at any time at least one replica must be defined and the second states that all *defined* replicas should have the same value. Note that when more than one replica is involved at different sites, the second part indicates that the constraint is distributed, and that a replica cannot be updated at one of its sites alone.

C is localized through $SC = C_1 \wedge \cdots \wedge C_N$, where

$$C_i = \begin{cases} x_i\uparrow \wedge (x_i = a), & \text{if } i \text{ has a replica,} \\ x_i\downarrow, & \text{otherwise.} \end{cases}$$

When node k (which already has a replica) is allowed to modify the replica, all other copies are invalidated leading to a transformed $SC' = C_1' \wedge \cdots \wedge C_N'$, where

$$C_i = \begin{cases} x_i\uparrow, & \text{if } i = k, \\ x_i\downarrow, & \text{otherwise.} \end{cases}$$

This transformation is effected correctly through relocalization. First, all nodes except k give up their read access changing from $x_i\uparrow \wedge (x_i = a)$ to $x_i\downarrow$; C is satisfied since x_k remains defined (see also remark 4 in section 4). Next, node k assumes write access changing from $x_k\uparrow \wedge (x_k = a)$ to $x_k\uparrow$; again C is satisfied since k is the only node with a defined replica (remark 3). Subsequently, k changes to read-only access by unilaterally changing to $x_k\uparrow \wedge (x_k = a)$, permitted since this implies $x_k\uparrow$ (remark 2). Finally other nodes such as i, given the value a, can change from $x_i\downarrow$ to $x_i\uparrow \wedge (x_i = a)$; C remains satisfied regardless of the order of those changes (remark 4).

Finally, we note that invalidation is not a necessary aspect of the caching idea. For example, consider the modulo-equality constraint E [12]:

$$E: \quad x_1 \equiv x_2 \pmod{k},$$

where x_1, x_2 are at nodes 1 and 2. E can be localized through $SC = E_1 \wedge E_2$:

$$E_i: \quad x_i = a \pmod{k}$$

and a is a constant. In other words, both local data items x_1, x_2 can be modified simultaneously and independently provided that $x_i \bmod k$ remains unchanged.

5.2. Leases

In the mobile context, we have seen that arbitrary disconnection can hold up nodes. One practical way of coping with this is to set a time limit on the validity of a replica obtained by a mobile node. This is the idea of data *lease*. A leased data item

is one shared by the requesters (leaseholders) each for a certain time interval. Typically, leaseholders have read-only access and are free to read the item (as long as the lease has not expired); in order to modify the data, however, a leaseholder must obtain permission from all other leaseholders, who give up their own read access when they give such permission.

Consider OFFER. It would be appropriate to give lease semantics to the replicated list of contracts so as to control sharing and yet not allow disconnection to hold up all others indefinitely. Many trucks can get the list in a read-only mode for a specified duration. When an operator wishes to make an offer, it gets an exclusive write access on the list and writes its own offer provided it betters existing offers (the initial offer is empty for each contract). For example (we will refer to this example twice later), let a truck k request a read lease on the current offer list. While viewing, it decides to make an offer and requests write access on the data. Having finished writing its offer ahead of the lease interval, it switches to read-only mode; other operators can now view its offer by getting read access. If one of these trucks after viewing the current offers wants to write its bid, it makes a requests for a write lease whereupon k gives up its read lease.

The global constraint is C as given above. Let a be the current value of the shared data X and T_i the valid lease duration for node i. C is localized through $SC = C_1 \wedge \cdots \wedge C_N$, where

$$C_i = \begin{cases} x_i\uparrow \wedge (x_i = a), \\ \quad \text{if } i \text{ has a valid lease for } T_i \text{ and time } t \in T_i, \\ x_i\downarrow, \quad \text{otherwise.} \end{cases}$$

To see how relocalization works, we will pursue a node k through the sequence followed by truck k in our OFFER example. Node k is initially a non-leaseholder, as it requests and gets a lease on X obtaining and creating a replica with the current value a for duration T_{k1}, later requests and gets permission to modify the replica for duration T_{k2} (it overlaps with T_{k1}), subsequently reverts to read-only access for duration T_{k3}, and finally is asked to surrender the replica. Its local constraint C_k goes through the sequence $C_k^0, C_k^1, C_k^2, C_k^3, C_k^4$ $(=C_k^0)$ and the other nodes modify their conditions so that C is satisfied at each step:

$$\begin{aligned} C_k^0 &= x_k\downarrow, \\ C_k^1 &= x_k\uparrow \wedge (x_k = a), \quad \text{for } t \in T_{k1}, \\ C_k^2 &= x_k\uparrow, \quad \text{for } t \in T_{k2}, \\ C_k^3 &= x_k\uparrow \wedge (x_k = b), \quad \text{for } t \in T_{k3}, \\ C_k^4 &= x_k\downarrow. \end{aligned}$$

Its initial constraint is C_k^0 with x_k undefined; at this time, some other node must have a lease on the data to satisfy C. Once it asks for a lease and gets a read-only lease, its constraint changes to C_k^1. No other node needs to change its local constraint. Later, when it wants to modify X, it asks every other leaseholder for permission. Node i gives permission along with the current value of X while changing its constraint C_i to $x_i\downarrow$. Suppose all leaseholders give permission. Then along with the last permission, node k changes its own constraint to C_k^2 allowing it to modify x_k. At this time, this is the only node with defined X. Suppose k finishes writing; it gives up its exclusive write access by changing to a read-only mode with constraint C_k^3 where b is the final value of x_k. Now node j requests a read lease and is allowed to set up its own constraint:

$$C_j = x_j\uparrow \wedge (x_j = b) \quad \text{for } t \in T_j.$$

Subsequently, when j wants a write lease, node k surrenders its access (unless its lease has expired already) by changing its constraint to $C_k^4 = C_k^0$. The correctness argument for this relocalization is the same as for the similar example using plain caching in section 5.1.

6. Localizing polynomial inequalities

Here we discuss the application of localization towards distributed polynomial inequality constraints. We make use of a geometric method.

Consider PICKUP. It 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. The Escrow method and the Demarcation Protocol have both shown how linear inequalities can be handled efficiently. But neither of them tell us how to handle the quadratic inequalities (owing to the product term $\mu_1\mu_2$, they cannot be converted into a linear form).

While illustrating the localization and relocalization approach using constraint P1 in section 4, we have already established that our approach can take care of the linear inequalities – basically in the same manner as the Escrow and Demarcation Protocol. But we can go further. We have been able to advance from linear to quadratic form by taking the localization perspective and asking how such constraints can be localized; the answer was a common approach – a geometric one – that provided solutions for both classes.

Any 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 ith coordinate for x_i. 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

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

The right-hand side of the above is C and the left-hand side 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 left-hand side 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 the subset is to $Dom(C)$, the better). Once we find such a rectangle, the node in charge

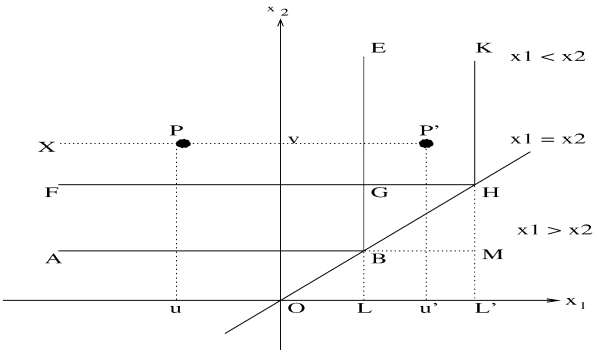

Figure 3. Managing a linear inequality.

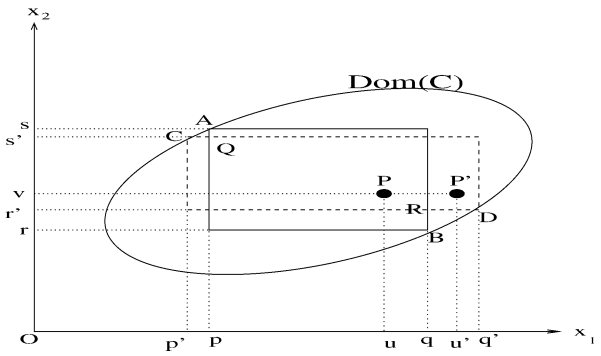

Figure 4. Managing a quadratic inequality.

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 intermediate rectangles are contained within $Dom(C)$. Thus, the geometric approach reduces to rectangle management. The initial rectangle can be empty; the rectangles can subsequently be set up through relocalization based on actual data. Here, we will not present the algorithms involved [15,16] but discuss two examples to illustrate the method.

For a simple example, consider $C = x_1 < x_2$, a linear inequality, where x_1 and x_2 reside at nodes 1 and 2, respectively. Geometrically, $Dom(C)$ corresponds to the half-plane above the line OH ($x_1 = x_2$) in figure 3. The rule

$$(x_1 < L) \wedge (L < x_2) \rightarrow (x_1 < x_2),$$

where L is a constant, allows us to localize C. The LHS (left-hand side) of the rule is the sufficient condition $SC = (x_1 < L) \wedge (L < x_2)$; geometrically, it is an open rectangle ABE inside the half-plane.

The current data $(x_1, x_2) = (u, v)$ is represented by P which is in rectangle ABE and therefore in the half-plane. If a local transaction at node 1 attempts to increase the value of x_1 to u' which is greater than L (for simplicity of exposition, let node 2 not concurrently change x_2), there is a violation of the local constraint but not of the global constraint (point P' is in the correct half-plane). Now the rectangle can be changed (relocalization) from ABE to, say, FHK via FGE as node 2 changes its bound to L' and then node 1 changes its bound to L'. Note that the change via AMK is not acceptable since that rectangle extends beyond the correct half-plane. Constraints P1 and P2 are of the same form as C and therefore the above description applies to them.

Now we will illustrate how the geometric approach works for 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 [21] we find that $Dom(C)$ is the interior of an ellipse. We then find a well-oriented rectangle (i.e., one with sides parallel to the x_1–x_2 coordinate system) inside the ellipse whose interior represents the rectangular subset we are seeking. Figure 4 shows such an ellipse containing a well-oriented rectangle with diagonal AB whose projections on the x_1 and x_2 axes give the local constraints $(p < x_1 < q)$ and $(r < x_2 < s)$.

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, 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 and its new projections on the axes x_1, x_2; these projections are the new local constraints. It first restricts its own bounds which it can do unilaterally (remark 2) thus shrinking the rectangle to QR and then informs node 1 that it can increase its bound enlarging the rectangle to CD.

7. From localization to PRO-MOTION

7.1. Implementation considerations

So far, we have taken all nodes to be equal: they were to communicate with each other in a peer-to-peer fashion. Owing to unpredictable and frequent disconnection, this becomes impractical: a hierarchical mode is preferable. The stationary server takes the role of a facilitator – all mobile hosts communicate directly with it (and indirectly with other MHs) because the server can be counted upon to remain connected almost all the time. Even better for the MH, the intermediate MSS layer (section 3) can talk to the server as a proxy on behalf of a MH that sends its request to the MSS and immediately disconnects. In the light of this arrangement, let us review our implementation of the lease.

- A MH makes its request to the compact manager of its associated server.
- The duration of the lease, i.e., a time limit on the validity of a replica obtained by a mobile node, is set by the server.
- In leases, local sufficient conditions hold only for the duration of the lease – thus local commits of updates on the MH are conditional.
- Before the lease expires, a writer should transmit its last value that can be safely committed within the lease period to the server and revert to a read-only mode for the rest of the lease unless it releases or re-negotiates it.

- A disconnected reader whose deadline has passed causes no headaches for other lease seekers.
- A disconnected writer k which was unable to communicate its final value before the deadline expired can be handled as follows.
  - The server would not contact k if it failed to transmit its final value. However, an optimization is to allow the server to request k to surrender an expired lease while granting a very small duration extension for k to transmit its final value and globally commit any locally committed updates.
  - If the server still elicits no response from k, the server can substitute for that missing value; thus other lease seekers will not be held up.
  - If no subsequent attempt has been made to get a write lease on the server-filled missing data, the final value obtained from k by the server through a later communication can be substituted despite the expired deadline (this is an example of optimistic concurrency control). For example, this would be applicable in BILLING. If another node did get a write lease in the interim, the server will notify k to redo its transactions.
- The special role of the server is very useful in the MANIFEST example. Ordinarily, the manifest data is read-only. Each truck gets a read lease on its own Q_i and this data is also maintained at the server. Thus, if a truck, let us say Truck 1, wants to change the quantity of material it loads, the server needs to calculate if the change can be handled not only in terms of the utility and economics of extra (or reduced) goods but also regarding the constraints P1–P4. Recall that Q_1 and Q_2 are both involved in the coefficients R1 and R2 in the constraints P1–P4. If the server can connect with and interrogate Truck 2, then the decision is easier. But even if it cannot, since Truck 2 only handles sufficient conditions, the server calculates if the parameters Q_2, μ_2, σ_2^2 assigned to Truck 2 are compatible with the new Q_1. If so, it allows Truck 1's request, thereby changing the constraints P1–P4 without Truck 2 being aware of it! This is an added benefit of localization.

7.2. Mapping to the compact

The compacts in PRO-MOTION are ideal for our approach. The local constraint C_i (obtained by localization) on data D for a MH i is directly mapped into a compact for D and handed over to the MH which becomes responsible for constraint enforcement. C_i is analyzed as follows: we split it into *temporal intervals* (if any), for each interval, check if *defined*, and if defined, infer *data changeability*, and *data restriction*. We encode the *temporal range* as Obligations (deadline or expiration time); if defined, Current_Status takes the value RW or RO based on *data changeability*, if it was never defined earlier, the value is NR (nonresident); but when changing from defined to undefined it takes the value stale, and the *data restriction* is encoded within Consistency_Rules.

Table 2
Transitions for the compact.

C_k^i	*CS*	Read()	Modify()	Commit()
C_k^0	NR	AskServer()	AskServer()	reject()
C_k^1	RO	return(val)	AskServer()	commit()
C_k^2	RW	return(val)	update()	commit()
C_k^3	RO	return(val)	AskServer()	if no update then commit() else AskServer()
C_k^4	stale	AskServer()	AskServer()	reject()

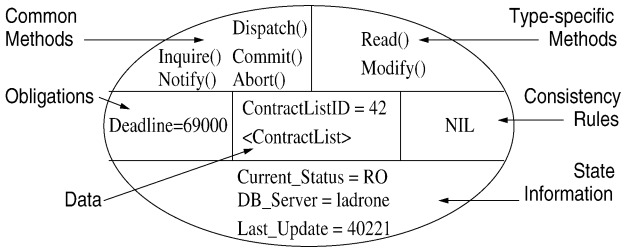

Figure 5. Compact: read-only leased data.

For illustration, we revert to the transitions involving truck k for the OFFER example described in section 5.2. Based on localization and relocalization, we can rest assured that if the compact for the replicated data (the list of contracts) has the requirements stipulated by C_k^0 through C_k^4, this truck can simply maintain these local constraints at each step and thereby satisfy the global constraint.

The compact includes, in addition to the common methods, two type-specific ones, Read() and Modify(). Table 2 lists for each constraint in the sequence, the corresponding compact entry for Current_Status (CS), as well as the functional description of the methods Read, Modify, and Commit.

For C_k^0, Current_Status is made NR (nonresident); Obligation and Consistency_Rules are empty since the data is undefined. Read (or Modify) results in an AskServer request to the (compact manager of the) server for a read lease.

Since the data is defined but unchangeable in C_k^1 (figure 5), Current_Status is made RO. The time range is translated into a deadline and entered in the Obligation field. The Consistency_rules field is left empty. Read returns the value and Commit performs a local commit. An invocation to Modify results in an AskServer request for write access.

For C_k^2, since the data is defined and changeable, Current_status becomes ReadWrite. Also, Modify now performs an update. Not shown in the table is the additional restriction captured by Consistency_Rules that each offer (time and dollar value) can only decrease the current offer.

C_k^3 is like C_k^1 except that Commit checks if the transaction has updated locally (in the RW period); if so, it asks the server if a global commit is possible.

Finally, the call to surrender the lease would be made by the server through Notify leading to the constraint reverting to C_k^4 which is like C_k^0 except that Current_Status is stale indicating a data value which may not be current (but in case it is, no data transfer will be necessary when it becomes defined later).

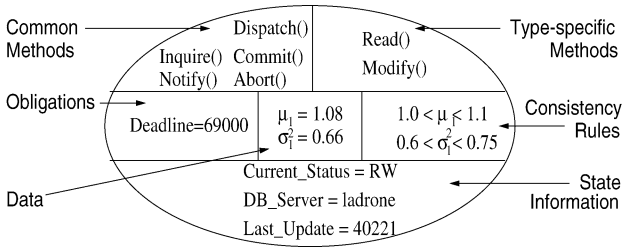

Figure 6. Compact for mean and variance.

7.2.1. The PICKUP constraints

Now, the quadratic inequality constraint in section 6 applies to P3 and P4 with one difference: P3, P4 involve four variables not two; consequently, our rectangle will be 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. Figure 6 shows the compact for one of the trucks. Along with Obligations and Current_Status, the Consistency_Rules field contains the bounds on μ_1, σ_1^2, actually the intersection of those obtained from all the four constraints.

While four variables are involved, μ_1 and σ_1^2 are on one MH 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., $0 \leqslant \sigma_i^2/\mu_i \leqslant L$, for $i = 1, 2$, where L is a constant. Using this, P3 and P4 reduce to the form of the quadratic constraint of section 6 based on two variables μ_1, μ_2 instead of four. Then figure 4 applies verbatim.

8. Conclusions

Exciting advances in wireless technology and semiconductors have thrown open the possibility of extending traditional database management functionality to the mobile computing environment. In this paper, we examined the problem of maintaining integrity constraints while executing transactions in a mobile environment and proposed a framework of localization. Our proposed framework is based on the reformulation of global constraints into a conjunction of local sufficient conditions and their dynamic maintenance. The consequent enhancement of autonomy of the mobile hosts empower them to avoid unbounded delays during constraint verification. This approach unifies techniques of maintaining replicated data with methods of enforcing polynomial inequalities. We discussed how this approach can be implemented in PRO-MOTION, a flexible infrastructure for transaction processing in a mobile environment. The method is to map the results of localization into the parameters of the compact which is the basic unit of data replication for caching, prefetching, and hoarding in PRO-MOTION.

Acknowledgements

Mazumdar and Chrysanthis are grateful to NSF for support under grants IRI-9509789 and IRI-95020091, respectively, and to their students as well as anonymous reviewers for their comments on the earlier workshop version of this paper.

References

[1] D. Barbará and H. Garcia-Molina, The demarcation protocol: A technique for maintaining linear arithmetic constraints in distributed database systems, in: *Proc. 3rd Internat. Conf. on Extending Database Technology* (1992) pp. 373–388.

[2] P.K. Chrysanthis, Transaction processing in a mobile computing environment, in: *Proc. IEEE Workshop on Advances in Parallel and Distributed Systems* (1993) pp. 77–82.

[3] P. Chrysanthis, G. Samaras and Y. Al-Houmaily, Recovery and performance of atomic commit protocols in distributed database systems, in: *Recovery in Database Management Systems*, eds. V. Kumar and M. Hsu (Prentice Hall, 1998).

[4] H. Garcia-Molina, Global consistency constraints considered harmful, in: *Proc. 1st Internat. Workshop on Interoperability in Multidatabase Systems* (1991) pp. 248–250.

[5] C.G. Gray and D. Cheriton, Leases: An efficient fault-tolerant mechanism for distributed file cache consistency, in: *Proc. 12th ACM Sympos. on Operating Systems Principles* (1989) pp. 202–210.

[6] B. Housel, G. Samaras and D. Lindquist, WebExpress: A client/intercept based system for optimizing web browsing in a wireless environment, Mobile Networks and Applications 3(4) (1998) 419–431.

[7] J. Ioannidis, D. Duchamp and G.Q. Maguire, IP-based protocols for mobile internetworking, in: *Proc. ACM SIGCOMM Sympos. on Communication, Architectures and Protocols* (1991) pp. 235–245.

[8] J. Kisler and M. Satyanarayanan, Disconnected operation in the Coda file system, ACM Transactions on Computer Systems 10(1) (1992) 13–25.

[9] N. Krishnakumar and A. Bernstein, High throughput Escrow algorithms for replicated databases, in: *Proc. 18th Internat. Conf. on Very Large Data Bases* (1992) pp. 175–186.

[10] A. Kumar and M. Stonebraker, Semantics-based transaction management techniques for replicated data, in: *Proc. ACM SIGMOD Internat. Conf. on Management of Data* (1988) pp. 117–125.

[11] J. Lloyd, *Foundations of Logic Programming* (Springer, Berlin, 1984).

[12] S. Mazumdar, Optimizing distributed integrity constraints, in: *Proc. 3rd Inernat. Sympos. on Database Systems for Advanced Applications* (1993) pp. 327–334.

[13] S. Mazumdar and P. Chrysanthis, Achieving consistency in mobile databases through localization in PRO-MOTION, in: *Proc. DEXA Internat. Workshop on Mobility in Databases and Distributed Systems* (1999) pp. 82–89.

[14] S. Mazumdar, M. Pietrzyk and P. Chrysanthis, Caching constrained mobile data, in: *Proc. 10th Internat. Conf. on Information and Knowledge Management* (2001) pp. 442–449.

[15] S. Mazumdar and G. Yuan, Localizing a class of distributed constraints: A geometric approach, J. Computing and Information (2000) 3/ICCI98/6/2.

[16] S. Mazumdar and G. Yuan, Localizing global constraints: A geometric approach, in: *Proc. 9th Internat. Conf. on Computing and Information* (1998) pp. 297–304.

[17] P.E. O'Neil, The Escrow transactional method, ACM TODS 11(4) (1986) 405–430.

[18] M. Pietrzyk, S. Mazumdar and R. Cline, Dynamic adjustment of localized constraints, in: *Proc. 10th Internat. Conf. on Database and Expert Systems Applications* (1999) pp. 791–801.

[19] E. Pitoura and G. Samaras, *Data Management for Mobile Computing* (Kluwer, Dordrecht, 1998).

[20] K. Ramamritham and P. Chrysanthis, A taxonomy of correctness criteria in database applications, VLDB J. 4(1) (1996) 181–293.

[21] L.L. Smail, *Analytic Geometry and Calculus* (Appleton-Century-Crofts, 1953).

[22] G.D. Walborn and P.K. Chrysanthis, Transaction processing in PRO-MOTION, in: *Proc. 14th ACM Annual Sympos. on Applied Computing* (1999) pp. 389–398.
[23] G.D. Walborn and P.K. Chrysanthis, PRO-MOTION: Support for mobile database access, J. of Personal Technologies 1(3) (1997) 171–181.

Subhasish Mazumdar is an Associate Professor of Computer Science at the New Mexico Insitute of Mining and Technology (New Mexico Tech). He received his B.Tech (Honors) and M.E. (Distinction) in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur and the Indian Institute of Science, Bangalore respectively; his M.S. and Ph.D. degrees in computer science were from the University of Massachusetts at Amherst. His current research interests include integrity of distributed and mobile databases, document management, and the use of conceptual modeling in software development. For his research he has received support from the National Science Foundation and the Sandia National Laboratory. He is a member of the ACM, IEEE Computer Society, and Sigma Xi.
E-mail: mazumdar@cs.nmt.edu

Panos K. Chrysanthis is a Professor of Computer Science at the University of Pittsburgh and an Adjunct Professor at Carnegie Mellon University. He received his B.S. from the University of Athens, Greece, in 1982 and his M.S. and Ph.D. from the University of Massachusetts at Amherst, in 1986 and 1991, respectively. His current research focus is on mobile and pervasive data management. In 1995, he was a recipient of the National Science Foundation CAREER Award for his investigation on the management of data for mobile and wireless computing. Besides journal and conference articles, his publications include a book and book chapters on advances in transaction processing and on consistency in distributed databases and multidatabases. He is currently an editor of the VLDB Journal, and was program chair of several workshops and conferences related to mobile computing. He is the ICDE 2004 Vice Chair for the area of distributed, parallel and mobile databases and the General Chair of MobiDE 2003 and Mobile Data Management 2005. He is a member of ACM (Sigmod, Sigmobile) and IEEE Computer Society.
E-mail: panos@cs.pitt.edu