Achieving Consistency in Mobile Databases through Localization in PRO-MOTION*

Subhasish Mazumdar
Department of Computer Science
New Mexico Tech
Socorro, NM 87801. USA.
mazumdar@cs.nmt.edu

Panos K. Chrysanthis
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260. USA.
panos@cs.pitt.edu

Abstract

There is great need and potential for traditional transaction support in a mobile computing environment. However, owing to the inherent limitations of mobile computing, we need to augment the well-developed techniques of Database Management Systems with new approaches. In this paper, we focus on the challenge of assuring data consistency. Our approach of localization is to reformulate global constraints so as to enhance the autonomy of the mobile hosts. We show how this approach unifies techniques of maintaining replicated data with methods of enforcing polynomial inequalities. We also discuss how localization can be implemented in PRO-MOTION, a flexible infrastructure for transaction processing in a mobile environment.

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 [14].

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 [3, 7]. 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) which 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 [2]. 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 first and second authors are grateful to NSF for support under grants IRI-9509789 and IRI-95020091 respectively and to anonymous reviewers for their thoughtful comments and suggestions.

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 (for set-based constraints, see [9]). 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, 15] for replicated data based on equality constraints, the *Escrow* method [12] and the *Demarcation Protocol* [1, 4] for linear inequality constraints; it has also enabled a hitherto unknown extension of Escrow to quadratic inequalities.

The conceptual framework of localization blends synergistically with the architecture of PRO-MOTION, a flexible infrastructure for transaction processing in a multi-tier, mobile client-server operating environment [17, 19]. 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*. The result of localization is simply a way of filling in certain components of appropriate compacts enabling unilateral commitment of local transactions. Since PRO-MOTION is a practical system under development (using Java), the localization approach is assured of realizability in a mobile environment.

The rest of the paper is structured as follows. In the following sections, we present a running example, introduce the PRO-MOTION infrastructure, explain the technique of localization, and outline how localization in our example would be handled in PRO-MOTION.

2. An Example

Mobile computers are becoming more and more common [18] in the trucking industry. 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 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. The driver of such a truck must come to the trucking company for paperwork, go to the source to pick up the material, and finally go to the destination to deliver them.

As a truck arrives at the trucking company, the driver is given a shipping manifest, a paper specification of the quantity of parts to be picked up plus pertinent information about the truck, the driver, and the source; on the basis of this paper, the driver will be allowed in at the source.

During the pick-up process, the goods are checked to see if they meet their specifications, i.e., if an attribute A (e.g., the 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. Therefore, 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. For our truck, however, there is an added complication: at the destination, the load from this truck will be merged with that from another source (via another truck), and the *overall* mean μ and variance σ^2 of A *of the merged collection* must be kept 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).

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

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.

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 variance σ_1^2, σ_2^2 respectively. Suppose the two trucks handle quantities Q_1, Q_2 respectively; then 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 P1 through P4 (two linear and two quadratic polynomial inequalities in four variables $\mu_1, \mu_2, \sigma_1^2, \sigma_2^2$) listed in Table 1.

So, our truck driver after measuring μ_1, σ_1^2, will attempt to verify the constraints P1 through P4. But to do so, it 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 quantity of material for the first and second truck are replicated on both their computers. The quantity can be changed on a mobile computer (e.g., if more

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

Table 1. PICKUP Constraints

goods are available and there is space in the truck) but only after the company's database has been consulted (both trucks delivering extra goods may not be acceptable) and the other truck is made aware of the change. Added complexity arises from the fact that any change in quantity Q_i affects R_1, R_2 and thus changes the constraints P1 through P4 for both trucks.

Clearly, disconnection during the change process can hold up the other truck's pickup process.

3. PRO-MOTION

In this section, we highlight the salient features of PRO-MOTION, a flexible infrastructure for transaction processing in a multi-tier, mobile client-server environment [17].

We assume a general mobile computing environment in which the network consists of stationary and mobile hosts (MHs) [6]. Certain specialized stationary hosts called *Mobility Support Stations* (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 or completely disconnected

The goal is to process as much of the transaction on the MH as possible, resorting to communication with the stationary database server 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 1). 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 that are invoked and controlled by the Transaction Manager. Compacts are supported at the stationary database server, the MSS, and the MH as follows.

- At the database server, 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 mobile host, 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 on the MH.

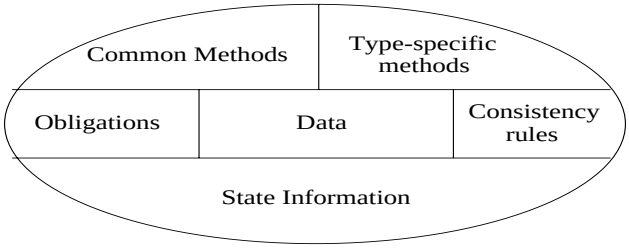

Figure 1. Compacts as Objects

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 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. 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 and, 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:

- inquire(), which retrieves useful information about the compact state (e.g., name, data type, version, cache status, free storage, and outstanding transaction IDs),
- dispatch(), which performs operations on the compact on behalf of transactions executing on the MH,

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

4. Localization

In this section, we elaborate on the notion of localization.

We assume that data is distributed among nodes $1, 2, ..., N$. A constraint C is *local* if it involves only one node and *distributed* otherwise. If there is a rule

$$C_1 \wedge C_2 \wedge ... \wedge C_N \rightarrow C,$$

such that for $1 \leq i \leq N$, C_i is local, then a distributed constraint C is said to be *localizable*. The variables and quantifiers (not shown) conform to the rules for Horn clauses [8].

We denote the left side of the rule by SC, a *sufficient condition* for C, and say that C is *localizable through* SC. So, instead of enforcing C which is distributed, we enforce local constraint C_i at node i for all i.

For example, let C be P1 (Table 1) which is distributed involving 2 nodes (the 2 trucks), i.e., $N = 2$, using 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 $R_1 L_1 + R_2 L_2 \geq M_0$, we see that P1 is localizable and that we can enforce $\mu_1 \geq L_1$, a local constraint, at the first truck, and $\mu_2 \geq L_2$ 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 modifiable to, say, SC' that could accommodate the updated value. SC' would be of the form:

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

Though the change 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. Thus, no synchronization of data should be necessary, i.e., a distributed transaction can be avoided.

SC is said to be *incrementally changeable* to SC' *through a sequence of constraints* $W_0(= SC), ..., W_i, ..., W_N(= SC')$ if, for $1 \leq i \leq n$, $W_i \rightarrow C$ and the sequence is incremental in the sense that, the difference between W_{i-1} and W_i is that exactly *one* conjunct C_j (for some j) in W_{i-1} is replaced by C'_j in W_i. Such a change of SC to SC' is referred to as *Incremental Update*.

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 $R_1 L_1 + R_2 L_2 = R_1 L'_1 + R_2 L'_2$; 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 incremental update through a sequence W_0, W_1, W_2 where

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

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

$$W_1 = (\mu_1 \geq L'_1) \wedge (\mu_2 \geq L_2)$$

may not have satisfied P1.

By *Localization*, we mean the substitution of the distributed constraint C by local constraints C_i at node i, and their dynamic adjustment through Incremental Update. The following remarks cover useful properties and clearly indicate why node autonomy is enhanced.

- If a local transaction at node i satisfies the local constraint C_i, no global constraint is checked and therefore unpredictable delays are avoided.
- Node i can unilaterally change its local constraint from C_i to C'_i if the new constraint is more restrictive, i.e., $C'_i \rightarrow C_i$, and the data which now satisfies C_i also satisfies C'_i. For example, the second truck could increase L_2 to L'_2 unilaterally.
- Incremental Update of SC can be done one node at a time possibly in a certain prescribed sequence. No distributed transaction with expensive commit protocols and distributed concurrency control is needed.

  Making sure that the nodes follow the sequence correctly and take care of simultaneous conflicting desires for constraint change is not trivial. Algorithms ensuring correct behavior under these conditions [10, 11] and experimental results [13] appear elsewhere.

5. Using Localization in PRO-MOTION

The compacts in PRO-MOTION are ideal for our approach. The local constraint C_i, the result of 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 obligation (deadline or expiration time); if defined, current_status takes the value `RW` or `RO` based on *data changeability*, otherwise, when changing from defined to undefined it takes the value `stale`, but if it was never defined earlier, the value is `NR` (nonresident); and the *data restriction* is encoded within consistency_rules. We will illustrate this below.

5.1. Handling Dynamic Replication

We have stated earlier that various data replication semantics such as *lease, callback, check-in/check-out* can be captured under the framework of localization and incremental update. For shortage of space, we will dwell only on the leasing technique [5].

Suppose a logical item X has replicas x_i at MH 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

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

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, the second part indicates that the constraint is distributed; and that a replica cannot be updated at one of its sites alone.

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 read access when they give permission. Before the lease expires, a writer communicates its final value to the server which subsequently forwards it to any requester and globally commits local updates. Below, we will not emphasize the special role of the server.

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 MH i. C is localized by finding a sufficient condition $SC = C_1 \wedge ... \wedge C_N$, where

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

To see how incremental update works, we will follow a MH k, which 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 $C_k^0, C_k^1, C_k^2, C_k^3, C_k^4 (= C_k^0)$:

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

Its initial constraint is C_k^0 with x_k undefined; at this time some other MH 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. Later, when it wants to modify X, it asks every other leaseholder MH for permission (a MH makes its request to the compact manager of the server it is associated with). MH 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, MH k changes its own constraint to C_k^2 allowing it to modify x_k. At this time, this is the only MH with defined X. Now MH j requests a standard lease (read access). MH k 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, and j is allowed to set up its own constraint as

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

Finally, suppose MH j wants to modify x_j. It requests MH k (via its server) to surrender the lease. MH k then changes its constraint to $C_k^4 = C_k^0$.

Suppose k had never received any request from j (or anyone else). Then, before k's write lease expired, it should have transmitted its last value that can be safely committed within the lease period to the server and reverted to a read-only mode for the remainining duration of the lease unless it released or re-negotiated it. 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 during which k could transmit its final value and globally commit any locally committed updates.

Consider MANIFEST. It would be appropriate to give the replicated data lease semantics. Most of the data items in the manifest are read-only. However, under certain conditions, a truck may want to change the quantity of material it loads. Let Truck 1 then go through the same sequence as MH k above. We will outline the effects on its compact.

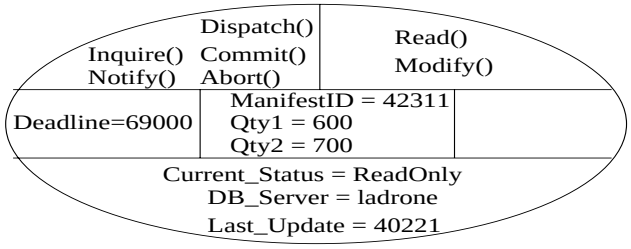

Figure 2. Compact: Read-only Leased Data

Based on localization and incremental update, we can rest assured that if the compact for the manifest data can be filled in with 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 methods for the common interface, two type-specific methods, Read() and Modify(). The following table 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.

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

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

The state of the compact for C_k^1 is shown in Figure 2. Since the data is defined but unchangeable, Current_Status is made RO (ReadOnly). The time range is translated into a deadline and entered in the Obligation field. The Consistency_rules field is left empty. 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 is made RW (ReadWrite); Commit is a local commit. C_k^3 is just like C_k^1 except that Commit checks if the transaction has updated locally (during 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 exactly like C_k^0 except that Current_Status is now stale indicating a data value which may not be current (but just in case it is, no data communication will be necessary when it becomes defined later).

Now let us consider the effects of disconnection in the mobile environment. Disconnections cannot cause delays in acquiring a lease. A reader whose deadline has passed and is disconnected causes no headaches for lease seekers. A writer which was unable to communicate its final value but the write lease interval has passed will not hold up the others (beyond the writer's lease interval) because the server will substitute for that missing value and proceed. However, in the latter case, there is a violation of the global constraint C: at the writer node, transactions committed locally (since the last communication) are not globally committed.

How can the constraint violation be repaired? One possibility is the following. The server would assign a default value (perhaps a NULL value) once the writer's deadline has passed. If no other attempt has been made to get a write lease on the data, the final value of MH k obtained through a later communication is stored despite the expired deadline. Otherwise, (if another MH did get a write lease), MH k will be told via Notify to redo its transactions.

In MANIFEST, only the quantity of goods Qty_1 would be the target of write requests from Truck 1. Recall that Qty_1 is also involved in the fractions R_1, R_2 which are coefficients of the polynomials in the constraints for PICKUP. Therefore the server would allow Truck 1 to modify Qty_1 only if the change is acceptable and Truck 2 has not yet picked up its goods. If Truck 1 becomes disconnected after getting a write lease, it cannot commit globally and will be forced to make sure that the PICKUP constraints are satisfied with the original Qty_1. While Truck 1 has a write lease on Qty_1, Truck 2 having given up its read lease on Qty_1 must wait to read it. If Truck 1's lease expires without communication, the server will tell Truck 2 to make its stale value current. Only if the second truck also fails to contact the server after giving up its read lease, will it have to wait.

5.2. Handling 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 through 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 incremental update 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.

Owing to our constraint reformulation perspective, we have been able to extend from linear to quadratic form through a common approach, a geometric one.

Any constraint $C = p(x_1, ..., x_N)$, 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. C can be geometrically interpreted as: *the datum* $(x_1, ..., x_N)$ *satisfies* C *if and only if the point* $(x_1, ..., x_N)$ *in the* N*-dimensional space is in* $Dom(C)$. Now, suppose we are able to find R_1, ..., R_N, each a range of $\mathcal{R}$ such that

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

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 MH 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. Incremental update 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. Here, we will not present the algorithms involved [10, 11] 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 MHs 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 (where L is a constant)

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

allows us to localize C. The LHS 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 MH 1 attempts to increase the value of x_1 to u' which is greater than L, 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 (incremental update) from ABE to, say, FHK via FGE as MH 2 changes its bound to L' and then MH 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.

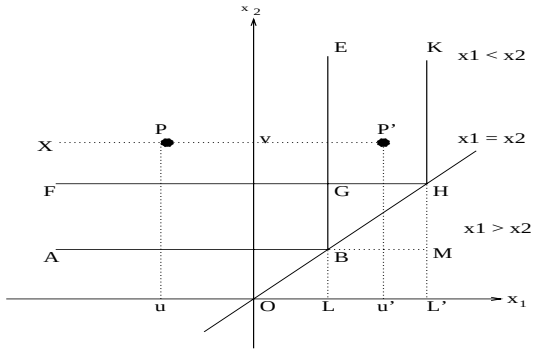

Figure 3. Managing a Linear Inequality

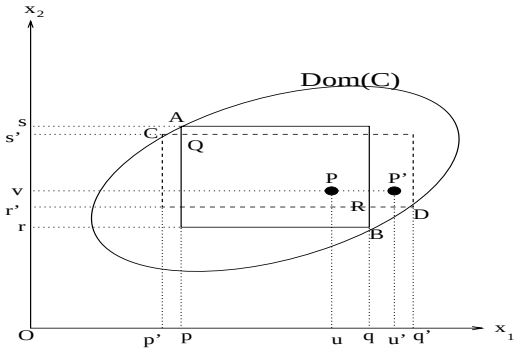

Figure 4. Managing the Interior of an Ellipse

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 [16] 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 MH 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. MH 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 (recall the properties of localization) thus shrinking the rectangle to QR and then informs MH 1 that it can increase its bound enlarging the rectangle to CD.

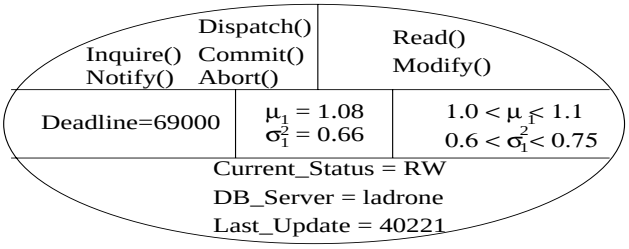

Figure 5. Compact for mean and variance

The above example 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 5 shows the compact for one of the trucks. Here, along with Obligations and Current_Status, we have an entry for Consistency_Rules: the bounds on μ_1, σ_1^2. These bounds are the intersection of those obtained from all the four constraints.

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.

6. 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 have examined the problem of maintaining data consistency 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 enhancing the autonomy of the mobile hosts. The mobile hosts, by enjoying greater autonomy, can avoid unbounded delays during constraint verification; also, the limitations of data replication caused by disconnections become clear. This approach unifies techniques of maintaining replicated data with methods of enforcing polynomial inequalities. We have 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.

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 Intl. Conf. on Extending Database Technology.*, pages 373–388, 1992.

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

[3] P. K. Chrysanthis. Transaction processing in a mobile computing environment. In *Proc. IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 77–82, 1993.

[4] H. Garcia-Molina. Global consistency constraints considered harmful. In *Proc. 1st Intl. Workshop on Interoperability in Multidatabase Systems*, pages 248–250, 1991.

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

[6] J. Ioannidis, D. Duchamp, and G. Q. Maguire. IP–based protocols for mobile internetworking. In *Proc. ACM SIGCOMM Symp. on Communication, Architectures and Protocols*, pages 235–245, 1991.

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

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

[9] S. Mazumdar. Optimizing distributed integrity constraints. In *Proc. 3rd Intl. Symp. on Database Systems for Advanced Applications.*, pages 327–334, 1993.

[10] S. Mazumdar and G. Yuan. Localizing a class of distributed constraints: A geometric approach. *Journal of Computing and Information*. To Appear.

[11] S. Mazumdar and G. Yuan. Localizing global constraints: A geometric approach. In *Proc. 9th Intl. Conf. on Computing and Information.*, 1998.

[12] P. E. O'Neil. The Escrow transactional method. *ACM Transactions on Database Systems*, 11(4):405–430, 1986.

[13] M. Pietrzyk, S. Mazumdar, and R. Cline. Dynamic adjustment of localized constraints. In *Proc. 10th Intl. Conf. on Database and Expert Systems Applications*, 1999.

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

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

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

[17] G. D. Walborn and P. Chrysanthis. PRO-MOTION: Management of mobile transactions. In *Proc. 11th ACM Annual Symposium on Applied Computing*, pages 171–181, 1997.

[18] G. D. Walborn and P. Chrysanthis. PRO-MOTION: Support for mobile database access. *Personal Technologies*, 1(3):171–181, 1997.

[19] G. D. Walborn and P. K. Chrysanthis. Transaction processing in PRO-MOTION. In *Proc. 14th ACM Annual Symposium on Applied Computing*, pages 389–398, 1999.