

PRO-MOTION: MANAGEMENT OF MOBILE TRANSACTIONS*

Gary D. Walborn
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
gwalborn@pitt.edu

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

Keywords: Data Caching, Mobile Computing, Transaction Processing, Semantics-based Concurrency Control.

ABSTRACT

In order to provide data consistency in the presence of failures and concurrency, database methods will continue to be important to the processing of shared information in a mobile computing environment. It is important, therefore, that we develop transaction processing systems that accommodate the limitations of mobile computing, such as frequent disconnection, limited battery life, low-bandwidth communication and reduced storage capacity, so that we can migrate existing database applications to mobile environments. In this paper, motivated by these needs, we propose a mobile transaction processing system that supports disconnected transaction processing in a mobile client-server environment. The proposed system employs *compacts*, which encapsulate access methods, state information and consistency constraints, to allow for local management of database transactions. Finally, we discuss the basic infrastructure needed to support this transaction processing system.

1 INTRODUCTION

Pentop and laptop computers are becoming smaller, lighter, more powerful, cheaper and easier to use. As these devices evolve, innovative new applications will be created to utilize their capabilities. A surge in the use of mobile computers as personal digital assistants (PDAs) is expected in the next few years [15]. The impact of PDAs has already been felt in some niche environments. Mobile computers are already being used by transportation companies, field maintenance teams and in telemedicine.

*This material is based upon work supported by the National Science Foundation under grants IRI-9210588 and IRI-95020091 and partially funded by B-Right Trucking Company.

In a mobile computing environment, the network is made up of *stationary* and *mobile* hosts (MHs) [16]. Unlike stationary hosts, MHs change location and network connections while computations are being processed. While in motion, MHs retain their network connection through the support of specialized stationary hosts with telecommunication ability, called *mobility support stations* (MSSs) (Figure 1). Each MSS is responsible for all of the MHs within a given geographical or logical area, known as a *cell*. When a MH leaves a cell serviced by an MSS, a *handoff* protocol is used to transfer the responsibility for support to the MSS of the new cell. This handoff may be as simple as establishing new communication links, or as complicated as migrating executing processes and database transactions in progress.

It is possible, even probable, that a mobile computer will become *disconnected* from the network due to (accidentally or intentionally) broken communication links. For example, in Figure 1, mh_1 becomes accidentally disconnected when entering a region out of the reach of any MSS. Many disconnections will be intentional (i.e., planned) or predicted. For instance, a weak radio link or a partially depleted battery may warn of impending disconnection. Also, mobile computers may enter an energy conservation mode (*sleep*), which resembles disconnection, in an attempt to extend battery life or to save on communication cost. Clearly, such a disconnection does not imply the failure of the disconnected machine. Today many mobile computers have enough memory, storage and processing ability to store a portion of the database and process transactions against that portion while disconnected. Once disconnected, a MH may reconnect after an indefinite time to an indeterminate stationary host.

To provide data consistency in the presence of failures and concurrency, database methods will continue to be important to the processing of shared information. It is important, therefore, that we find transaction processing models which accommodate the limitations of mobile computing, such as frequent disconnection, limited battery life, low-bandwidth communication and reduced storage capacity. Operations on shared data must insure correctness of transactions executed on both the stationary hosts and MHs. Blocking of transactions which are executed on either the stationary or mobile hosts must be minimized and autonomy of the MH should be maximized to reduce communications and increase

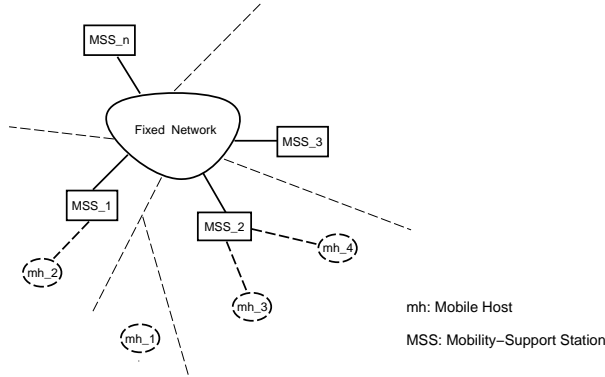


Figure 1: Mobile Network

concurrency on both mobile and stationary hosts.

Recent research has attempted to develop appropriate transaction models for mobile computing that aim to support transactions which perform updates at the mobile computer [9, 23]. These efforts propose new mobile transaction models and correctness criteria for data consistency that are weaker than the standard *serializability* [7] so that they can cope more effectively with the restrictions of mobility and wireless communication. Even though serializability might be too strong as a correctness criterion for a number of applications, there are important applications, including existing business applications such as inventory databases [20], that require the data consistency guarantees offered by serializability.

Our goal has been to devise methods to ensure serializability and traditional ACID transaction properties¹ for mobile database applications, despite the various limitations introduced by mobility and portability. To this end, we have revisited and extended *escrow methods* [22] and certain caching strategies [18, 11] to support disconnected database operations in a mobile environment in which data are moved between the stationary servers and mobile clients. Our transaction processing model, called PRO-MOTION², employs *compacts* to provide support for both dynamic replication for *escrowable* items with improved caching techniques for *non-escrowable* items. This combination of dynamic replication and caching techniques allows competing transactions executing on mobile and stationary clients to share data items without blocking (and with minimum coordination) while maintaining integrity constraints [32]. Transactions executing on a disconnected mobile host are allowed to commit locally (i.e., unilaterally), permitting mobile database clients to execute competing transactions on cached and replicated data items, incorporating the modifications into the database when reconnection occurs.

In the next section, we present a real-world scenario which demonstrates some of the various requirements of mobile transaction processing. In Section 3, we

¹ACID: atomicity, consistency, isolation and durability

²PRO-MOTION stands for the “Pro-active Management of Mobile Transactions”. In addition, the title serves to indicate that the system is designed to provide support for mobility, i.e., motion.

describe our approach to mobile transaction processing, introduce the notion of *compacts* and discuss the advantages they provide. In Section 4, we discuss how the proposed PRO-MOTION system can be used to support our motivating application. Finally, in Section 5, we present our conclusions and suggest areas for further inquiries and testing.

2 MOTIVATING APPLICATION

Mobile computers are becoming more and more common in the trucking industry. Each truck is fitted with a small computer which communicates via satellite or radio links to a central site managed by the trucking company. The mobile computer runs specialized software which gathers data from vehicle instruments, digitizer pens and keyboard. The collected data is used to update the corporate database and is used for billing, compliance, equipment management and driver payroll. Often electronic means will be used to transmit funds directly to the driver for fuel, tolls, permits, living expenses and repairs.

In order to better understand the requirements of mobile transaction processing, let us look at a typical scenario faced by a contract carrier which has accepted a contract to move a large quantity of fertilizer from the manufacturing facility to a number of rural farms and co-ops. If the trucks are privately owned and sub-contracted to the trucking company, each driver is free to accept or reject any load that is offered. Even though the company has little control over each individual truck, the company will be penalized if it fails to complete delivery of the fertilizer within a specified period.

Many interesting problems are raised here. For example, each truck should be offered a portion of the available loads, but care should be taken so that the quantity of fertilizer offered to all of the trucks does not greatly exceed the quantity available. If, for example, every truck is offered the entire quantity, trucks could be stranded with insufficient loads to pay for expenses. In this case, the total amount of fertilizer allocated to each truck may be allowed to vary from actual availability (controlled divergence), as long as the total variance is within limits and eventual consistency is obtained. Since a cached copy at the mobile host may carry sufficient information to make an attempted pickup, connection to the centralized computer is not required before proceeding to the factory for a load.

As trucks arrive to load with fertilizer, each truck must obtain a shipping manifest from the trucking company. Each manifest uniquely identifies a specific load of fertilizer and indicates the location and time of loading, the pertinent information about the truck and driver, the exact measure of material loaded and a description of any exceptions that should be noted for insurance purposes. This information should be made permanent in the centralized database before the truck is permitted to travel or receive advances for fuel. Therefore, it may be necessary to be connected to successfully complete this transaction.

Once the load is delivered, the driver records the delivery date and time, obtains a signature from the receiving party, notes any discrepancies between the delivered goods and the original bill of lading and checks the list of available loads in order to proceed to the next shipper for loading. The delivery information should be incorporated in the centralized database as soon as possible, but a delay will only postpone the billing process. With the proper cache control and reconciliation, this transaction could be finalized locally, in spite of disconnection, but made permanent in the centralized database whenever reconnection occurs.

In this scenario, one can easily identify three distinct types of transactions with respect to data consistency requirements:

- transactions which tolerate controlled divergence, with the possible existence of a global constraint (i.e., total offers do not exceed available fertilizer +/- tolerance),
- a traditional (ACID) transaction which must be made permanent in the database before any other transactions pertaining to the load may be processed (i.e., recording load information and obtaining unique manifest number) and,
- an update of manifest information which should be made permanent in the database “as soon as possible”, but need not delay the acquisition and loading of more material (i.e., eventual consistency).

The transaction processing system presented in the balance of this paper will provide the mechanisms to support these different types of transactions which maintain database consistency while allowing flexibility in coordinating and reconciling competing transactions.

3 PRO-MOTION: A MOBILE TRANSACTION PROCESSING SYSTEM

The limitations of the mobile environment present a number of challenges to traditional transaction processing systems. To that end, in this section, we propose a new transaction processing system, called PRO-MOTION, to deal with the problems introduced by disconnection and limited resources. The salient features of PRO-MOTION are:

- the use of *compacts*, which function as the basic unit of caching and control,
- exploitation of object semantics wherever possible to improve site autonomy and increase concurrency and, finally,
- the introduction of a transaction management subsystem, (consisting of a *compact manager* at the database server, a *compact agent* at the mobile host and a *mobility manager* executing at the MSS) to negotiate and manage compacts and provide local transaction management for the MH.

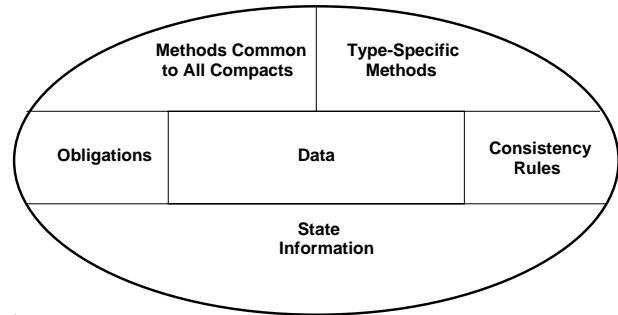


Figure 2: Compacts as Objects

These features which are described below, when used together, can minimize the handicap imposed by mobile limitations.

3.1 Compacts

As mentioned above, our mobile database system architecture assumes that there is a database server which resides on one or more stationary hosts and that mobile applications executing on the MHs operate on data managed by this database server. In a traditional client-server environment, each transaction executing on the MH would send requests for data and updates directly to the database server. If the MH is currently connected to the network, this is expensive because each operation requires a set of messages, placing a heavy demand on batteries and bandwidth. If the mobile happens to be disconnected, it is impossible, prohibiting the processing of transactions during disconnection. This clearly indicates the need to support database operations locally on the MHs.

In PRO-MOTION, we propose to support the caching of data on the MHs for access to mobile applications by means of *compacts*. A compact is, broadly speaking, a satisfied request to cache data, enhanced with *obligations* (such as a deadline), *restrictions* (such as a set of allowable operations) and *state information* (such as the number of accesses to the object). The compact represents an agreement between the database server and the mobile host. In this agreement, the database server delegates control of the data to the MH. The MH, in return, agrees to assume responsibility for the data and to honor specific conditions set forth by the database server. As a result, the database server need not be aware of individual operations executed by individual transactions on the MH but, rather, sees periodic updates to a compact for each of the data items manipulated by the mobile transactions.

Compacts are represented in our system as objects (Figure 2) which encapsulate

- the cached data,
- methods (i.e., code) for the access of the cached data,
- information about the current state of the compact,

- consistency rules, if any, which must be followed to guarantee global consistency of the data item,
- obligations, such as a *deadline* which creates a bound on the time for which the rights to a resource are held by the mobile host, and
- methods which provide an interface with which the MH may manage the compact.

The management of compacts is a cooperative effort by the database server and the mobile hosts. Compacts are obtained from the database via *requests* by the MH when a real or anticipated demand is created. The compacts are periodically *updated* as the result of processing transactions on the MH. 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 store.

Whenever a MH needs data, the MH sends a request for the data to the database server. If data is available to satisfy the request, the database server creates a compact which is transmitted to the MH to provide the data and methods to satisfy the needs of transactions executing on the MH. The request can be tailored to cause only the transmission of missing or outdated components of a compact. In this way, 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 is recorded in a *compact registry* which is used by the compact agent to track the location and status of all open compacts.

Each compact has a common interface which is used by the compact agent to manage the compacts listed in the compact registry. The basic set of methods necessary to manage compacts includes,

- **inquire**, so that the compact agent can obtain the current status of the compact,
- **notify**, so that the compact can receive notification when status of the MH changes,
- **dispatch**, used to process operations on the compact issued by transactions executing on the MH,
- **commit**, to make the operations of a specified transaction permanent on the database, and
- **abort**, to abandon the changes made to the compact data by a given transaction.

The implementation of a common interface simplifies the design of the compact agent and guarantees the minimum acceptable functionality of a specific compact instance.

The flexibility offered by compacts allows PROMOTION to support a number of dynamic replication schemes (as well as caching) with a variety of consistency constraints. Compacts can be used to represent both simple schemes such as check-in/check-out items

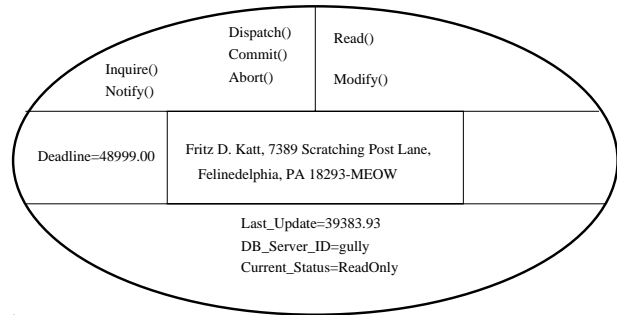


Figure 3: Leases as Compacts

and leases [11], as well as other more complex correctness criteria [26]. Let us illustrate this by showing how leases can be realized using compacts.

Lease semantics provide that a shared data item is given to all requesters (leaseholders) with an expiration time. Leaseholders are free to read the item (as long as the lease has not expired), but must obtain permission from all other leaseholders before modifying a leased item. Therefore, a compact designed to support leases (Figure 3) includes, in addition to the methods for the common interface, two type-specific methods, **read** and **modify**. The **read** method simply checks the expiration (specified in the compact as an obligation) and, if unexpired, satisfies the read request, returning the value of the compact data (Fritz...). When the **modify** method is invoked, the compact must communicate with the database server. The database server obtains permission from the other leaseholder compacts and communicates permission (or refusal) to the lease compact on the MH performing the modification. A lease compact requires no consistency rules and contains a single obligation, the expiration time (**Deadline=48999.00**). The compact state (**Last_Update**) is maintained by the type-specific and interface methods.

Because compacts include the code for access to shared objects, compacts provide a high degree of adaptability which may be exploited to respond dynamically to changes in the mobile environment. For example, compacts may react to an increased availability of hard disk by automatically increasing the amount of data cached or adapt to an overloaded communication subsystem by reducing the frequency of transmitted updates. The use of compacts, therefore, creates a flexible and adaptable framework to support mobile transaction processing.

In the next section, we elaborate further on the usage of compacts and illustrate one additional advantage offered by compact, namely, their ability to support concurrency control methods which exploit object structure and/or semantics [4, 3, 12, 10].

3.2 Using Compacts to Support Semantics-Based Concurrency Control

In order to better support disconnected operations and strict data consistency, such as that provided by serializability, the mobile transaction processing system

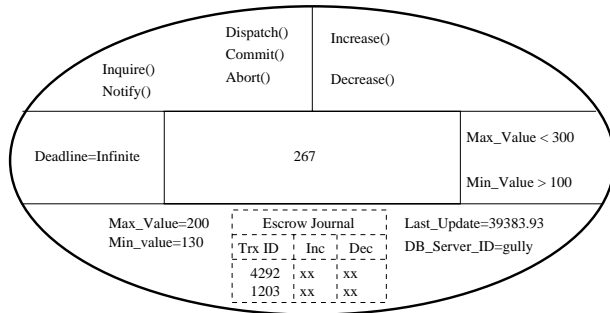


Figure 4: Escrows as Compacts

can exploit object semantic information to provide finer granularity of caching and concurrency control and to allow for asynchronous manipulation of the cached objects and unilateral commitment of transactions on the mobile host. As an example, escrow items and fragmentable items have characteristics that make them especially suitable for mobile transactions.

The basic idea behind both *escrow* and *fragmentable* items [32] is to split large or complex objects into smaller fragments of the same type by exploiting the object organization. With the appropriate split, a mobile host can cache an *data partition* (consisting of one or more fragments) of just the right size, minimizing the storage requirements on a mobile host. The second idea is to make these fragments the unit of reconciliation of updates, that is, the unit of consistency. To allow more flexibility (as well as to deal with situations in which fragmentation under strict consistency requirements is not possible) applications can explicitly define the consistency constraints to be enforced.

A “master copy” of the escrowable or fragmentable data resides on a database server. Mobile hosts specify the granularity of the data to be cached when placing a compact request by specifying the required size of the data partition. The data partition is logically removed from the “master copy”, packaged into a compact and transmitted to the MH. The data contained in the compact is only accessible by the transactions on the mobile host. However, the remaining part of the “master copy” is not affected and it is available to other MHs as subsequent compacts. Object fragments can be *logical* (i.e., escrow) or *physical* (i.e., fragments) divisions of the data object. During the update, physical fragments need to be physically re-assembled into a single object while logical fragments are combined with some logical or arithmetic operation.

In order to support unilateral commitment of transaction executing on a mobile host, we must retain the effects of transaction operations on each fragment when the fragments are merged. The consistency conditions embodied in the compact methods specify constraints on the fragment which need to be satisfied to maintain the consistency of the entire object. These conditions might include allowable operations and constraints on their input values and conditions on the state of the object. Some operations on fragmentable items may be

disallowed or restricted to guarantee that the fragments may be properly merged.

Figure 4 illustrates a compact designed to support escrowable data (i.e., a portion of an aggregate³ item which has been allocated to satisfy requests from transactions executing on the mobile host). In addition to the methods which provide a common interface, this compact contains two type-specific methods, *increase* and *decrease*. The data (i.e., 267) may only be accessed by these two methods. Each call to *increase* is validated against the maximum value (i.e., 300) specified in the consistency rules, and the compact state (i.e., Escrow Journal), which reflects validated requests by concurrent transactions. The call succeeds only if the consistency rules will not be violated. Similarly, each call to *decrease* is validated against the minimum value (i.e., 100) and the compact state. Adherence to the consistency rules will insure that the global consistency of the aggregate values will be guaranteed even if transactions are allowed to commit unilaterally on the MH. Because unilateral commitment is possible, transactions on other MHs may proceed even though the only obligation, a deadline for return of the escrow quantity, has been set to unlimited (i.e., Infinite).

3.3 Infrastructure

As mentioned in the introduction, one of our motivations has been to support existing database applications by facilitating data access by transactions on mobile hosts. In such an environment, it might not be easy, or even possible, to incorporate logic to manage compacts into the legacy database server⁴. If a database server lacks compact management capabilities, a *compact manager* can provide that functionality. The compact manager acts as a front-end to a database server, shielding the database server from the idiosyncrasies of the mobile environment [27]. The compact manager may execute on an independent host, or it may execute on the same host as the database server.

If a compact manager is added to a legacy database, our system utilizes an open nested transaction model as the basis for concurrency control and recovery for mobile transactions processed against the database server. To the database server, the compact manager appears to be an ordinary database client, executing a single, large, long-lived transaction. This single, large transaction becomes the root of our nested transaction. Resources needed to create compacts are obtained by this transaction through normal database operations (reads and writes). Mobile transactions (transactions processed by each mobile host) appear as children in the open nested transaction. The transactions processed on the mobile host appear as siblings. Each sibling transaction may commit or abort independently as long as the consistency constraints expressed in the compacts are not violated.

³An aggregate is a data objects which represents a quantity of identical and interchangeable items, such as bushels of wheat or dollars in an advance account.

⁴A similar situation exists in multidatabase systems that attempt to integrate pre-existing database systems while retaining the *design autonomy* of the component database systems [8].

The responsibility for the correct execution of mobile transactions is assumed by the MH and accomplished by utilizing the methods encapsulated in the compacts. The root transaction is managed by the database server and committed by the compact manager.

On each MH, a *compact agent* is responsible for processing requests on behalf of transactions executing on the MH. These compact agents are more than just the interface between the compact manager and the transactions on the mobile hosts. The compact agent is much like the daemon responsible for cache management in the CODA file system [18]. The compact agent handles disconnections and manages storage on a MH. It monitors activity and interacts with the user and applications to maintain lists of items which are candidates for caching. However, unlike the CODA daemon, or other cache managers [6, 29], the compact agent is actively involved in *transaction processing* on the mobile host, acting as a transaction manager for transactions executing on the mobile host. The compact agent is responsible for concurrency control, logging and recovery. Consequently, transaction requests, commits, aborts and journals are managed by the compact agent. Requests from local transactions are processed against the compacts and are granted or denied. When a transaction commits or aborts, the compacts and transaction journals are updated accordingly.

Each compact includes a set of methods used for management which are common to all compacts. In addition, each compact may contain specialized methods which support the particular type of data or concurrency control specific to that particular compact. As a result, many of the functions associated with the compact agent are actually executed by the compacts themselves upon receipt of messages from the compact agent triggered by executing transactions or changes in the MH state.

At strategic intervals, the compact's state at the compact manager is updated to reflect the effects of all outstanding committed transactions. The specific triggers for updates may vary, but should include:

1. When renegotiation must be performed for additional resources and the update can be *piggybacked* onto the request.
2. As part of a handoff procedure when a transfer to a new cell is initiated.
3. When the number of commit requests reaches a predetermined threshold, usually determined by the memory capacity of the mobile host.
4. In a "panic" situation arising from impending disconnection from:
 - weak or low battery power,
 - deliberate disconnection by the user,
 - or partial loss or weakness of signal detected by the communication subsystem.
5. A "safe" interval before an approaching deadline. (e.g., a trigger could be set for midway between the last attempt and the deadline,
6. When specifically requested by a critical application being processed on the mobile host.

Once the update is acknowledged by the compact manager, the compact agent updates journals and logs appropriately. In this manner, entire groups of committed transactions may be processed with a single update to the compact at each end, with significant savings in communication overhead.

Remember that each of the interactions between the compact agent and the compact manager are processed via the MSS. To improve system performance, the MSS can take an active role in the processing of compacts between the compact agent and the compact manager. The module which performs these functions on the MSS is called the *mobility manager* (MM). Once an update is sent to the MM, the MM functions on behalf of the compact agent to complete delivery of the update message. This *update by proxy* helps insure that the updates are received by the compact manager in a timely fashion. In case of disconnection, the update can be recorded by the compact manager and the acknowledgment can be stored by the MM and properly processed later, once the MH reconnects. The MM maintains *mobility tables* in which each *mobile control block* (MCB) contains location and database access information which pertains to a single MH. When a MH moves between cells, the MM uses the information stored in the MCB to facilitate the handoff to a new MSS (and corresponding MM).

The addition of the compact manager, compact agent and mobility manager provides a functional infrastructure that will move some of the responsibility for processing and committing transactions down to the MSS and the MH, reducing the dependence on communications with the database server.

4 DISCUSSION

Let's examine how the proposed transaction processing system based on compacts can be used to permit the various types of transactions set forth in the motivating application (Section 2). To recap, the requirements were:

- to make the quantity of fertilizer available to all of the trucks, while preventing gross overcommitment,
- the recording of manifest information before proceeding, and
- the eventual logging of delivery information, concurrent with the beginning of the next load.

The first transaction type is well suited to escrow operations. The quantity of fertilizer represents an aggregate item which can be roughly distributed among the mobile hosts (trucks) so that each truck would have some idea of the amount of fertilizer remaining to be moved without reconnecting to the centralized computer. The deadline could be set so that the truck would have to

obtain periodic updates reflecting the amount of material remaining. Eventually, the deadline might be set according to the actual *contract* deadline, if no loads will be available after that point. Renegotiation of compacts would be used to allow aggressive drivers to control the bulk of material to be hauled.

In the case of the second transaction, a compact would be used to obtain exclusive access to the manifest. Perhaps a small set of manifests would be reserved to each truck and kept in a single compact. The deadline would be arranged so that if the unit disappeared, the manifests would eventually return to the centralized system to be allocated to other trucks. No use of a manifest from an expired compact would be allowed to satisfy the transaction, as that manifest may have already been re-assigned to another load. An indefinite deadline would reserve the manifest to the requesting truck forever. It would require some type of administrative intervention to return the manifest to the central database.

In the final type of transaction, the assigned manifest is held by the truck for update. Since the manifest is the sole property of the truck holding the compact, the update can be applied locally to the compact and updated in the central database when reconnection occurs. If the compact has expired, there are two possibilities:

- If no other transaction has modified the manifest, the update may be stored despite the expired deadline. This is equivalent to optimistic concurrency control procedures.
- If the copy of the manifest in the centralized database *has* been modified, the update will be aborted and the truck driver will be requested to repeat the proof of delivery.

The *notify* method in this particular compact could be written to reflect this activity.

In each of these cases, we have processed as much of the transaction on the mobile host as possible, without resorting to communication with the centralized database. The central database is contacted only when convenient or when absolutely required by the semantics of the transaction. The compacts associated with each transaction type are designed to meet the requirements of that particular transaction, simplifying the job of local transaction management.

5 CONCLUSIONS

PRO-MOTION allows for improved transaction processing by disconnected mobile hosts without grossly impairing access by transactions executed by the stationary host and other MHs. Furthermore, these techniques allow for automatic recovery of resources held by MHs which exceed negotiated deadlines for reconnection. Since each compact encapsulates access methods with data, the mobile will automatically receive code necessary to manipulate data in the compact. This eliminates the need to write a comprehensive local transaction manager which contains code for types which may never be

used and allows for the automatic updating of access methods as the system evolves.

Commit processing in our system is handled by a single round of update messages with an associated acknowledgment. Consolidation of multiple update messages reduces messages even further. The consolidation of multiple updates into a single message by the compacts should also provide for improved scalability and reduced message traffic when compared to “flat” methods. Even though additional overhead is required to transmit and manage the compacts, we believe that these methods will reduce *overall* message traffic and improve autonomy while maintaining data consistency. Because compacts include the code necessary to manage the associated data, the size of each transfer is larger than sending the data alone, but the overall traffic is much less than what would be required to manage the same data at the database server. More importantly, compacts provide the means to build a flexible and adaptable support system for the management of mobile transactions.

We are building PRO-MOTION to test these techniques and experiment with various implementations. The common interface has been specified and the design of the compact agent is partially completed. Also, we have developed various topologies that we would like to investigate further [31].

References

- [1] Alonso R., D. Barbara and H. Garcia-Molina. Data Caching Issues in an Information Retrieval Systems. *ACM Transactions on Database Systems*, 15(3):359–384, Sept. 1990.
- [2] Acharya A. and B.R. Badrinath. Delivering Multicast Messages in Networks with Mobile Hosts. *Proc. of the 13th Int’l Conf. on Distributed Computing Systems*, pp. 292–300, 1993.
- [3] Agrawal D., A. El Abbadi and A. K. Singh. Consistency and Orderability: Semantics-Based. *ACM Transactions on Database Systems*, 18(3):460–486, Sep. 1993.
- [4] Badrinath B. R. and K. Ramamritham. Semantics-based Concurrency Control: Beyond Commutativity. *ACM Transactions on Database Systems*, 17(1):163–199, Mar. 1992.
- [5] Barbara D. and H. Garcia-Molina. The Demarcation Protocol: A Technique for Maintaining Constraints in Distributed Database Systems. *Proc. of the Int’l Conf. on Extending Data Base Technology*, pp. 377–392 Mar. 1992.
- [6] Barbara D. and T. Imieliński. Sleepers and Workaholics: Caching Strategies in Mobile Environment. *Proc. of the 1994 ACM SIGMOD Int’l Conf. on Management of Data*, pp. 1–12, May 1994.
- [7] Bernstein P. A., V. Hadzilacos and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 1987.
- [8] Breitbart Y., H. Garcia-Molina and A. Silberschatz. Overview of Multidatabase Transaction Management. *VLDB Journal*, 1(2):181–293, Apr. 1992.

- [9] Chrysanthos P. K. Transaction Processing in a Mobile Computing Environment. *IEEE Workshop on Advances in Parallel and Distributed Systems*, pp. 77–82, Oct. 1993.
- [10] Chrysanthos P. K., S. Raghuram and K. Ramamritham. Extracting Concurrency from Objects: A Methodology. *Proc. of the 1991 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 108–117, May 1991.
- [11] Gray C. G. and D. Cheriton. Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. *CS Technical Report CS-TR-90-1298*, Stanford University, Jan. 1990.
- [12] Herlihy M. P. and W. Weihl. Hybrid Concurrency Control for Abstract Data Types. *Proc. of the 7th ACM SIGACT-SIGMOD-SIGART Sym. on Principles of Database Systems*, pp. 201–210, Mar. 1988.
- [13] Huang Y., P. Sistla and O. Wolfson. Data Replication for Mobile Computers. *Proc. of the 1994 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 13–24, May 1994.
- [14] Imieliński T. and B. R. Badrinath. Querying in Highly Mobile Distributed Environment. *Proc. of the 18th Conf. on VLDB*, pp. 41–52, Aug. 1992.
- [15] Imieliński T. and B. R. Badrinath. Mobile Wireless Computing: Challenges in Data Management. *Communication of ACM*, 37(10):18–28, Oct. 1994.
- [16] Ioannidis J., D. Duchamp and G. Q. Maguire. IP-Based protocols for mobile internetworking. *Proc. of ACM SIGCOMM Sym. on Communication, Architectures and Protocols*, pp. 235–245, 1991.
- [17] Jain R. and K. Narayanan. Network Support for Personal Information Services to PCS Users. Preprint: IEEE Conf. Networks for Personal Communications, Mar. 1994.
- [18] Kisler J. and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3-25, 1992.
- [19] Krishnakumar N. and A. Bernstein. High Throughput Escrow Algorithms for Replicated Databases. *Proc. of the 18th Conf. on VLDB*, pp. 175–186, Aug. 1992.
- [20] Krishnakumar N. and R. Jain. Protocols for maintaining inventory databases and user service profiles in mobile sales applications. *Proc. of the Mobidata Workshop*, Nov. 1994.
- [21] Kumar A. and M. Stonebraker. Semantics-based Transaction Management Techniques for Replicated Data. *Proc. of the 1988 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 379–388, May 1988.
- [22] O'Neil P. The Escrow Transactional Method. *ACM Transactions on Database Systems*, 11(4):405–430, Dec. 1986.
- [23] Pitoura E. and B. Bhargava. Maintaining Consistency of Data in Mobile Distributed Environments. *Proc. of 15th Int'l Conf. on Distributed Computing Systems*, pp. 404–414, June 1995.
- [24] Pu C. and A. Leff. Replica Control in Distributed Systems: An Asynchronous Approach. *Proc. of the 1991 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 377–386, May 1991.
- [25] Ramamritham K. Real-Time Databases. *Int'l Journal of Distributed and Parallel Databases*, 1(2):199–226, Apr. 1993.
- [26] Ramamritham K. and P. K. Chrysanthos. A Taxonomy of Correctness Criteria in Database Applications. *VLDB Journal*, 4(1):181–293, Jan. 1996.
- [27] Seal K. and S. Singh. Loss Profiles: A Quality of Service Measure in Mobile Computing. Submitted for publication.
- [28] Soparkar N. and A. Silberschatz. Data-value Partitioning and Virtual Messages. *Proc. of the 9th ACM SIGACT-SIGMOD Sym. on Principles of Database Systems*, pp. 357–367, 1990.
- [29] Tait D. C. and D. Duchamp. Service Interface and Replica Management Algorithm for Mobile File System Clients. *Proc. of the 1st Int'l Conf. on Parallel and Distributed Information Systems*, pp. 190–197, 1991.
- [30] Yeo L. H. and A. Zaslavsky. Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment. *Proc. of the 14th Int'l Conf. on Distributed Computing Systems*, June 1994.
- [31] Walborn G. and P. K. Chrysanthos. An Escrow Method to Support Disconnected Mobile Database Operations. *CS Technical Report 95-09*, University of Pittsburgh, Feb. 1995.
- [32] Walborn G. and P. K. Chrysanthos. Supporting Semantics-Based Transaction Processing in Mobile Database Applications. *Proc. of the 11th Symp. of Reliable Distributed Systems*, pp. 31–40, Sept. 1995.