Transaction Processing in Mobile, Heterogeneous Database Systems

James B. Lim and A.R. Hurson

Abstract—As technological advances are made in software and hardware, the feasibility of accessing information "any time, anywhere" is becoming a reality. Furthermore, the diversity and amount of information available to a given user is increasing at a rapid rate. In a mobile computing environment, a potentially large number of users may simultaneously access the global data; therefore, there is a need to provide a means to allow concurrent management of transactions. Current multidatabase concurrency control schemes do not address the limited bandwidth and frequent disconnection associated with wireless networks. This paper proposes a new hierarchical concurrency control algorithm. The proposed concurrency control algorithm—v-lock—uses global locking tables created with semantic information contained within the hierarchy. The locking tables are used to serialize global transactions, detect and remove global deadlocks. Additionally, data replication, at the mobile unit, is used to limit the effects of the restrictions imposed by a mobile environment. The replicated data provides additional availability in case of a weak connection or disconnection. Current research has concentrated on page and file-based caching or replication schemes to address the availability and consistency issues in a mobile environment. In a mobile, multidatabase environment, local autonomy restrictions prevent the use of a page or file-based data replication scheme. This paper proposes a new data replication scheme to address the limited bandwidth and local autonomy restrictions. Queries and the associated data are cached at the mobile unit as a complete object. Consistency is maintained by using a parity-based invalidation scheme. A simple prefetching scheme is used in conjunction with caching to further improve the effectiveness of the proposed scheme. Finally, a simulator was developed to evaluate the performance of the proposed algorithms. The simulation results are presented and discussed.

Index Terms—Mobile computing environment, global information sharing process, concurrency control, caching and prefetching, simulation and analysis.

1 Introduction

The conventional notion of timely and reliable access to global information sources is rapidly changing. Users have become much more demanding in that they desire and sometimes even require access to information "any time, anywhere." The extensive diversity in the range of information that is accessible to a user at any given time is also growing at a rapid rate. Furthermore, rapidly expanding technology is making available a wide breadth of devices with different memory, storage, network, power, and display requirements to access this diverse data set.

Current multidatabase systems (MDBS) are designed to allow timely and reliable access to large amounts of heterogeneous data from different data sources in an environment that is characterized as "sometime, somewhere." Within the scope of these systems, multidatabase researchers have addressed issues such as autonomy, heterogeneity, transaction management, concurrency control, transparency, and query resolution [6], [7], [9], [14], [18]. These solutions were based upon fixed clients and servers connected over a reliable network infrastructure. However, the concept of *mobility*, where a user accesses data through a remote connection with a portable device, has introduced additional complexities and restrictions in a multidatabase system. These include: 1) a reduced capacity network connection, 2) processing and resource restrictions, and 3) effectively locating and accessing information from a multitude of sources. An MDBS with such additional restrictions is called a mobile data access system (MDAS).

Nevertheless, regardless of the hardware device, connection medium, location of data, and type of data accessed, all users share the same requirements: timely and reliable access to various types of data that can be classified as:

Private data, i.e., personal schedules, phone numbers, etc. The reader of this type of data is the sole owner/user of the data.

Public data, i.e., news, weather, traffic information, flight information, etc. This type of data is maintained by one source and shared by many.

Shared data, i.e., a group data, replicated or fragmented data of a database. A node actually may contribute to maintaining consistency and participate in distributed decision making with this type of data.

Traditionally, in a distributed environment, to achieve higher performance and throughput, transactions are interleaved and executed concurrently [28]. Concurrent execution of transactions should be coordinated such that there is no interference among them. In an MDAS environment, the concurrent execution of transactions is a more difficult task to control than in distributed database systems. This is mainly due to the inferences among global transactions, inferences among global and local transactions, local autonomy of each site, and frequent network

- *J.B. Lim is with MJL Technology, MJL Building, 204-5 Nonhyun-1 Dong, Kangnam-ku, Seoul 135-826, South Korea. E-mail: jblim@mjl.com.*
- *A.R. Hurson is with the Department of Computer Science and Engineering, 220 Pond Lab., The Pennsylvania State University, University Park, PA 16802. E-mail: hurson@cse.psu.edu.*

Manuscript received 19 Aug. 1998; revised 3 Mar. 2001; accepted 10 July 2001.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 107298.

disconnection. Furthermore, to mitigate the effects of weak communications or disconnection, some form of data duplication should be used, at the mobile unit, to provide additional availability in case of a weak connection or disconnection.

Researchers have extensively studied caching and replication schemes that may be used to address the constraints of a wireless link [6], [9], [21], [36], [42]. Current distributed database replication schemes are not suitable for an MDAS environment because consistency cannot be effectively maintained due to local autonomy requirements and communication constraints. The proposed caching schemes in distributed databases are also not suitable for an MDAS environment. Due to the autonomy requirements of the local DBMS, the local information about the validity of a page or file in the local system is not available globally. Accordingly, any type of invalidation, polling, or time-stamp-based method would be too impractical and inefficient to use and, in many cases, impossible.

In this paper, a new hierarchical concurrency control algorithm is introduced to reduce the required communication overhead. The proposed scheme offers higher overall throughput and faster response times—timely access to data. The concurrency control for global transactions is performed at the global level in a hierarchical, distributed manner. A hierarchical structure was chosen for several reasons:

1. A hierarchical organization offers potentially higher performance in that processing and data structures can be easily distributed.
2. The reliability of the system is increased by eliminating a single point of failure for the MDBS involved in a global transaction.
3. The algorithm is designed to work with the Summary Schemas Model (SSM), which is a distributed, hierarchical, multidatabase environment [10]. The proposed algorithm is easily integrated into the SSM.

Concurrent execution of transactions is accomplished by creating global locking tables using semantic information within the hierarchy of the SSM.

This work also addresses the limited bandwidth and local autonomy restrictions of an MDAS by using *automated queued queries* (AQ^2) and caching of data in the form of a *bundled query* (BUNQ). An AQ^2 is a form of prefetching that preloads data onto the mobile unit to allow the user to have access to the required data in case of a disconnection. A bundled query is an object that consists of a query and its associated data. Read-only queries are cached as a bundled query and are validated using a simple parity-checking scheme. Guaranteeing write consistency while disconnected is extremely difficult or impossible due to local autonomy requirements in an MDAS. Consequently, any transactions containing write operations are directly submitted to the MDAS system or are queued during disconnection and submitted upon reconnection. The proposed caching and prefetching algorithm reduces the access latency and provides timely access to data, notwithstanding the limited bandwidth restrictions imposed by an MDAS environment.

The remainder of this paper is organized into several sections. Section 2 addresses the necessary background material on mobile systems and multidatabase systems, concurrency control, and data replication. A new computing environment in which wireless-mobile computing elements are superimposed on a multidatabase system is also introduced. In Section 3, the proposed multidatabase concurrency control algorithm is introduced and discussed. Additionally, this section addresses the proposed data duplication model and presents protocols for maintaining consistency in the MDAS environment. In Section 4, a simulation of the proposed concurrency control scheme and the data duplication scheme is introduced. A simulation model is developed to test the effectiveness of the proposed schemes. Subsequently, the results are presented and analyzed. Finally, Section 5 concludes the paper and gives future directions.

2 Background

Accessing a large amount of data over a limited capability network connection involves two general aspects: 1) the mobile networking environment and 2) mobility issues. The mobile environment includes the physical network architecture and access devices. Mobility issues include adaptability to a mobile environment, autonomy, and heterogeneity.

A mobile application must be able to adapt to changing conditions [41]. These changes include the network environment and resources available to the application. A balance between self-autonomy and dependence on a server for a mobile user is very critical. A resource-scarce mobile system is better served by relying upon a server. However, frequent network disconnection, limited network bandwidth, and power restrictions argue for some degree of autonomy. As the environment changes, the application must adapt to the level of support required from stationary systems (servers).

The concept of mobility implies a much more diverse range and amount of accessible data. Subsequently, a remote access system must provide access to a larger set of heterogeneous data sources. Moreover, it must be able to facilitate the bandwidth, resource, and power restrictions of mobile systems. Multidatabase systems share many of the same characteristics as mobile systems.

2.1 A New Computational Environment

A multidatabase system (MDBS) provides a logical integrated view and method to access multiple local database systems while hiding the hardware and software intricacies from the user. Access to the local DBMS through a much more diverse and restrictive communication and access devices is the natural extension to a traditional MDBS environment, i.e., an MDAS environment [31]. The following section briefly introduces this environment and discusses various issues involved in providing multidatabase functionality to a mobile user. The current SSM model is extended to incorporate data replication for mobile units in an MDAS environment. The interested reader is referred to [10], [31] for a more detailed discussion of the SSM and an MDAS environment.

2.1.1 Multidatabase Environment

A multidatabase system is a global system layer that allows distributed access to multiple preexisting local database systems. One of the essential features of a multidatabase system is local site autonomy that comes in the form of

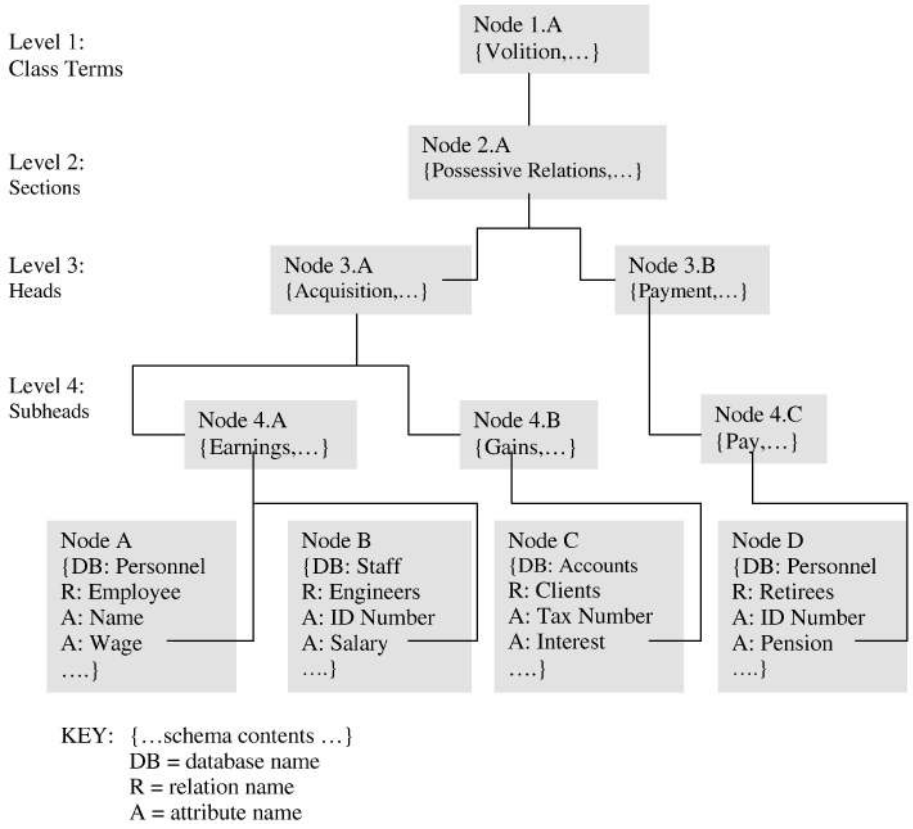


Fig. 1. Sample summary schemas metadata.

Design Autonomy, *Communication Autonomy*, and *Execution Autonomy*. When joining an MDBS, a local DBMS should not require any software or hardware changes—each local database management system (DBMS) maintains complete control over local data and resources.

In addition to autonomy, heterogeneity is also an important aspect of a multidatabase system. Support for heterogeneity is a tradeoff between developing and making changes in both hardware and software and limiting participation. Consequently, as the number of systems and the degree of heterogeneity among these systems increases, the cost of integration into the global MDBS increases. In an MDBS, autonomy and heterogeneity leads to issues in schema integration, query languages, query processing, and transaction management [9].

2.1.2 Summary Schemas Model for Multidatabase Systems

Accessing data in a heterogeneous system is a challenging problem. Multidatabase language and global schema systems suffer from inefficiencies and scalability problems. The summary schemas model (SSM) has been proposed as an efficient means to access data in a heterogeneous multidatabase environment [10]. It acts as a backbone to a multidatabase for query resolution. The identification of the terms that are semantically similar is one of the key concepts in SSM. Terms in language can be related with each other through semantic relationships like synonyms, hypernyms, hyponyms and antonyms. SSM uses the taxonomy of the English language that contains at least hypernym/hyponym, and synonym links among terms to build a hierarchical metadata. This hierarchical meta structure provides an incrementally concise view of the data in the form of summary schemas. The SSM hierarchy (Fig. 1) consists of leaf-node schema and summary schemas. A leaf node schema represents an actual database, while a summary schema gives an abstract view of the information available at the schemas of its children. The hypernyms of terms in the children of a particular SSM node form the summary schema of that node. Since hypernyms are more general or abstract than their hyponyms, many terms could map into a common hypernym. This reduces the overall memory requirements of the SSM metadata as compared to the global schema approach. As can be noted from Fig. 1, terms "Salary, Wage, Interest, and Pension" are integrated into various hypernyms at different levels of the SSM hierarchy and ultimately, all are semantically represented as "Volition" at the root of the hierarchy. The semantic distance metric (SDM) between the hypernym and their respective hyponyms are relatively small; hence, even though each summary schema does not contain exact information of its children but only an abstracted version, it preserves the semantic contents of its children—the summary schema nodes provide a more concise view of the data by summarizing and integrating the schemas of its children nodes. Each local access term needs to be mapped to an entry-level term in the underlying linguistic taxonomy. This mapping is represented in a data-structure called the Local-Global-Schema (LGS). This structure contains pairwise mapping between a local access term (could be cryptic) and the leaf level term in the linguistic taxonomy. The LGS is implemented semiautomatically by the local DBA(s). The procedure of populating the SSM hierarchy after the creation of LGS is automated.

The SSM intelligently resolves user queries [10]. A goal of a multidatabase system is to allow the user to extract information desired from distributed heterogeneous databases transparently—i.e., Data-Location Transparency, Data-Representation Transparency, and Data-Distribution Transparency. The user of SSM could issue either imprecise or precise queries. The imprecise queries are those that may have access term(s) different from the local access term(s) and/or may not specify the location(s) of the data. A precise query, on the other hand, uses the exact local-access terms and has the location of the database specified. A precise query is resolved by querying the specified database. The imprecise query resolution is more involved in that the semantic intent of the user query has to be identified and, based on that intention, the query will be resolved.

The hierarchical structure of the SSM is used to resolve imprecise queries. The query resolution starts at the node issuing the query. Each term "*a*" in the query is compared with the terms in the schema "*s*" at that node. If the SDM(s) between all query terms and schema terms are less than or equal to the specified semantic distance metric, the query is resolved either at that node (if local node) or in the children of that node (if an SSM node). On the other hand, if "*a*" does not match with "*s*," that is, "*a*" and "*s*" within the boundary of the specified semantic distance, are not linguistically related, the search proceeds to the parent of the current node. This process recursively continues until either the search reaches the top of the hierarchy with no downward search path or it reaches the local-node where the search is resolved or the search fails at a particular node on a **downward traversal.** The latter condition arises when the query terms do not match the schema terms at a particular node. Resolution of a global query allows the translation of a query into the corresponding subqueries and identification of a global coordinator in charge of global concurrency control for that query.

Definition 1. *A global coordinator of a query is the lowest summary schema node that semantically contains the information space requested by the query.*

A simulation and benchmark of the SSM model was performed and the benefits were shown in [10], [16]. The simulator compared and contrasted the costs of querying a multidatabase system using precise queries (access terms and locations are known) against the intelligent query processing capability of the SSM for the imprecise queries. The results showed that the intelligent query processing (imprecise query) of the SSM and an exact query (precise query) incurred very comparable costs (i.e., there were only small overhead costs involved) [10]. Interestingly, using intelligent SSM query processing actually outperformed an exact query in certain circumstances [16]. The SSM provides several benefits to the traditional multidatabase systems that can be directly applied to an MDAS. These include:

- The SSM provides global access to data without requiring precise knowledge of local access terms or local views. The system can intelligently process a user's query in his/her own terms.
- The SSM's hierarchical structure of hypernym/hyponym relationships produces incrementally concise views of the global data. The overall memory requirements for the SSM hierarchical metadata, compared to the requirements of a global schema, are drastically reduced by up to 94 percent [16]. Subsequently, the SSM's metadata could be kept in main memory, thus reducing the access time and query processing time. Furthermore, for very resource limited devices in an MDAS, only portions of the upper levels of the SSM metadata structure could be stored locally, which would still provide a global (albeit less detailed) view of the data.
- The SSM can be used to browse/view the global data. The user can either 1) follow semantic links in a summary schemas node or 2) query the system for terms that are similar to the user's access terms. In either case, the SSM could be used to browse data by "stepping" through the hierarchy, or view semantically similar data through queries.

An SSM prototype in C++ on the Solaris platform, through a project funded by the US Defense Advanced Research Projects Agency, was developed to identify the practicality and feasibility of the SSM approach. The prototype is based on the client/server model and is a totally distributed system. The prototype is capable of performing both imprecise and precise queries successfully. To demonstrate the capability of the SSM in handling heterogeneous data sources, the prototype was tested using two Oracle databases—1) flight databases and 2) steam boiler databases. The SSM hierarchy was created on Sparcstation, Alpha, and Indy platforms. The user queries, both precise and imprecise queries, were submitted from a PC. A graphical user interface (GUI) was developed to facilitate user communication. In addition, a translator capable of translating the GUI to SQL query was developed. The prototype demonstrated the functionality and feasibility of SSM. The prototype also showed the bottlenecks of the system, for instance, in some cases, we found a heavy amount of communication in the SSM hierarchy. In addition, it was found that, based on the underlying application databases, the development of application-specific thesaurus would improve accuracy and performance of the system.

Based on the experiences gained and to reduce the communication bottleneck, a decision was made to enhance the scope of the prototype by application of the mobile software agent technology. At present, an SSM user is able to initiate queries in the form of software agents. Our preliminary performance analysis has shown that, in many instances, transferring computation (software mobile agent) rather than massive data, across the network reduces the network traffic.

2.1.3 The MDAS Environment

In addition to the issues involved in an MDBS, accessing a large amount of data over a network connection of an MDAS involves the mobile networking environment and mobility issues. Overall, the main differentiating feature between an MDAS and an MDBS is the connection of servers and/or clients through a wireless environment and the devices used to access the data. However, both environments are intended to provide timely and reliable access to the globally shared data.

A summary of the issues facing a multidatabase and mobile system is given in Table 1. Due to the similarities in the objectives of effectively accessing data in a multidatabase and

TABLE 1
Comparative Analysis of Multidatabase and Mobile System

System Issues	Multidatabase	Mobile System
Heterogeneous Interoperability	Hardware and Software heterogeneity	Identical to an MDBS
Transaction Management and Concurrency Control	Correct transaction management should satisfy the ACID properties (Atomicity, Consistency, Isolation, and Durability)	Identical to an MDBS
Distribution Transparency	Distribution of the data is transparent to the user.	Identical to an MDBS
System Transparency	The user should be able to access the data irrespective of the system	Identical to an MDBS
Representation Transparency	Representation transparency includes naming differences, format differences, structural differences, and missing or conflicting data.	Identical to an MDBS
Intelligent Search and Browsing of Data	The system should provide a means for the user to efficiently search and browse the available data.	Identical to an MDBS
Intelligent Query Resolution	The system should be able to efficiently process and optimize a query submitted to the user	Identical to an MDBS
Location Transparency	Location of the data is transparent to the user.	Identical to an MDBS
Site Autonomy	Local control over resources and data. Three different forms of autonomy include design, communication and execution.	Local control over resources and data. The degree of autonomy required depends upon the degree of mobile support offered by the system.
Location Dependence	N/A	The content of the data is physically dependent upon the location of the user and/or system.
Disconnect and Weak Connection Support	N/A	The system should provide a means to access the data while faced with a disconnection or weak connection
Support for Resource Scarce Systems	N/A	The system should address the inherent limitations of various resource scarce access devices. These include processing, storage, power, and display limitations.

a wireless-mobile computing environment, we propose to superimpose a wireless-mobile computing environment on an MDBS [31]. We call this new system a mobile data access system (MDAS). By superimposing an MDBS onto a mobile computing environment, one should be able to easily map many solutions from one environment to another. In Sections 2.2 and 2.3, two of these issues—concurrency control and data duplication—are examined in more detail.

2.2 Transaction Management and Concurrency Control

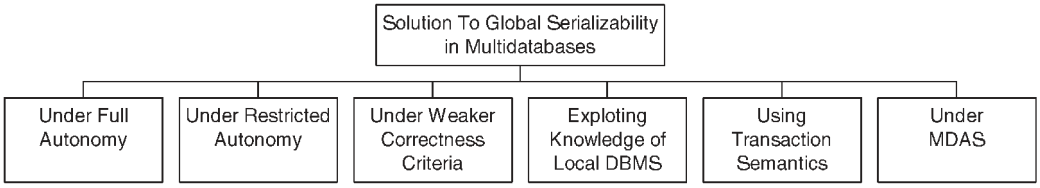
In a distributed environment, concurrency is used as a means to increase the throughput and to reduce the response time. Data access in an MDBS is accomplished through *transactions*. Concurrency control involves coordinating the operations of multiple transactions that operate in parallel and access shared data. By interleaving the operations in such a manner, the potential of interference between transactions arises. The concurrent execution of transactions is considered correct when the ACID properties (**Atomicity**, **Consistency**, **Isolation**, and **Durability**) are hold for each individual transaction [26]. Maintaining the ACID properties is desirable in any MDBS. The autonomy requirement of local databases in an MDAS (multidatabase system) introduces additional complexities in maintaining serializable histories because the local transactions are not visible at the global level. Consequently, the operations in a transaction can be subjected to large delays, frequent or unnecessary aborts, inconsistency, and deadlock. Two types of conflicts may arise due to the concurrent execution of transactions—*direct* and *indirect* conflicts.

Definition 2. *A direct conflict between two transactions* $\mathrm{T_a}$ *and* $\mathrm{T_b}$ *exists if and only if an operation of* $\mathrm{T_a}$ *on data item* x *(denoted* $\mathrm{o(T_a}(x))$*) is followed by* $\mathrm{o(T_b}(x))$*, where* T_a *does not commit or abort before* $\mathrm{o(T_b}(x))$ *and either* $\mathrm{o(T_a}(x))$ *or* $\mathrm{o(T_b}(x))$ *is a write operation.*

Definition 3. *An indirect conflict between the two transactions* $\mathrm{T_a}$ *and* $\mathrm{T_b}$ *exists if and only if there exists a sequence of transactions* $\mathrm{T_1, T_2, \ldots T_n}$ *such that* $\mathrm{T_a}$ *is in direct conflict with* T1, T1 *is in direct conflict with* $\mathrm{T_2, \ldots;}$, *and* $\mathrm{T_n}$ *is in direct with* $\mathrm{T_b}$ *[7].*

An MDAS should maintain a globally serializable history for correct execution of concurrent transactions. This means that the global history should be conflict-free while preserving as much local autonomy as possible. While the MDBS is responsible for producing a globally serializable history, it is assumed that the local concurrency control system will produce a locally serializable history as well. It is important to note that the MDBS needs to address both direct and indirect conflicts between global transactions. Table 2 summarizes several concurrency control algorithms that have been advanced in the literature [43].

TABLE 2
A Taxonomy for Different Global Serializability Schemes



2.3 Data Replication for Weak Connections and Disconnection

The communication limitations of an MDAS environment, i.e., restricted bandwidth, weak connections, and disconnection (either voluntary or involuntary), may require that a portion of the data in some form be made readily available to the mobile unit. A large amount of related work has been reported in the area of distributed file systems [30], [40], [42], distributed replication [3], [22], [29], [37], and distributed/web caches [12], [14], [27]. The local autonomy requirement of an MDAS environment does not lend itself to the direct application of these works. There are two main objectives in maintaining replicated data in a mobile or distributed environment:

- *Data Consistency*. The duplicate copies of data should be somehow consistent within the system. Current schemes do not maintain transactional (ACID) consistency. Several research models address the connectivity issues related to a wireless environment. These include weak and lazy consistency models that relax the consistency requirements of a system and allow the existence of inconsistent copies of data [1], [36], [37]—operations are performed on a conditional or delayed basis. These models eventually find discrepancies among the replicated copies of data and attempt to reconcile the differences by either updating the information or semantically compensating for the discrepancy.
- *Data Access*. When bandwidth limitations or disconnection occurs, the impact on the availability of the data to the mobile user should be minimized as much as possible. Our work addresses the ability to process data in a transactional sense in an MDAS environment—i.e., read and write operations are guaranteed to be consistent while connected to the system, and read-only operations are allowed during disconnection.

2.3.1 Data Replication

Replication schemes differ slightly from caching schemes in that the replicated data is accessible by other systems outside of the system on which the data resides. There are two types of general replication schemes: 1) primary/secondary copy (PSC) replication [34], [35] and 2) voting-based replication schemes [4], [27]. Depending upon the PSC replication scheme, read operations may access data that is inconsistent with the primary copy. PSC replication does not work in an MDAS environment because write operations to replicated data do not reach the primary copy when disconnected. Therefore, write consistency is not guaranteed. In a distributed system, data consistency is guaranteed with voting based replications; however, they tend to be more expensive and require much more communication. In an MDAS environment, the local replica, while disconnected, cannot participate in the decision/voting process and any changes to the local data may result in an inconsistency.

2.3.2 Data Caching

Caching is an effective means of data duplication that is used to reduce the latency of read and write operations on data. Early research on the disconnected operation was done with file system based projects, e.g., the Coda file system [42]. When a disconnection occurs, a cache manager services all file system requests from the cache contents. Coda logs all transactions during periods of disconnection and replays the log upon reconnection to synchronize the two copies of data. Ficus [39], Little Work [28], Bayou [17], and Odyssey [41] are some of the projects that support the disconnect operation.

File or page-based caching schemes use invalidation protocols in order to maintain data consistency. The mobile unit and server keep track of cached data—each time the data item changes, the server notifies the mobile unit. Alternatively, to reduce the amount of communication, the mobile unit can check for invalidated data only when accessing data. As with replication, the invalidation of data is not possible when disconnected and, thus, consistency cannot be guaranteed.

Web-based caching is becoming more important as information on the Internet continues its rapid expansion. The two most common forms of cache consistency mechanisms used on the Internet include time-to-live fields (TTL) and client polling [12].

There is currently no replication or cache-based scheme for mobile distributed systems that guarantees consistency. The disconnected operation of a mobile system and the local autonomy requirements of the local systems make consistency management extremely difficult. Some schemes try to use the concept of compensating transactions (or operations) to keep replicated data consistent. However, compensating transactions are difficult to implement and, in some cases, a compensating transaction cannot semantically undo the transaction (or operation) [7].

3 Concurrency Control and Data Replication for an MDAS

Within the scope of the MDAS environment, information sources can be either stationary or mobile [21]. In addition, a user request could be either directed toward public data or shared data. Issues regarding accessing public data are beyond the scope of this work and the interested reader is referred to [13]. The proposed concurrency control algorithm

is defined on an environment in which information sources are stationary and user requests are aimed at shared data sources. Under these conditions, the v-locking algorithm is intended to reduce the amount of communication and, hence, to reduce the effect of frequent disconnections in wireless environment.

3.1 Concurrency Control Scheme

The proposed v-locking algorithm uses a global locking scheme (GLS) to serialize conflicting operations of global transactions. Global locking tables are used to lock data items involved in a global transaction in accordance with the two-phase locking (2PL) rules. In typical multidatabase systems, maintaining a global locking table would require communication of information from the local site to the global transaction manager (GTM) regarding locked data items. In an MDAS environment, this is impractical due to the delay, amount of communication overhead, and frequent disconnections involved. The MDAS software is distributed in a hierarchical structure similar to the hierarchical structure of the SSM (see Section 2.1.2). Subsequently, transaction management is performed at the global level in a hierarchical, distributed manner.

The motivation behind using a hierarchical transaction management organization was due to the hierarchical structure of the SSM and the fact that such an organization offers higher performance and reliability [10], [32]. A global transaction is submitted at any node in the hierarchy—either at a local node or at a summary schema node. The transaction is resolved, mapped into subtransactions, and its global transaction manager is determined by the SSM structure [10].

Our concurrency control algorithm is based upon the following assumptions:

1. There is no distinction between local and global transactions at the local level.
2. A local site is completely isolated from other local sites.
3. Each local system ensures local serializability and freedom from local deadlocks.
4. A local database may abort any transaction at any time within the constraints of a distributed atomic commit protocol. The most widely supported distributed atomic commit protocol in commercial systems is the two-phase commit (2PC) and, therefore, our algorithm relies on the constraints of the 2PC. A distributed atomic commit protocol usually means that the local system will have to give up a certain degree of autonomy. In the case of the 2PC, once the "Yes" vote is given in response to a "Prepare to Commit" message, the local system may not subsequently abort any operation involved with the vote.
5. Information pertaining to the type of concurrency control used at the local site will be available. In order for systems to provide robust concurrency and consistency, in most systems, a *strict* history is produced through the use of a strict 2PL scheme. Therefore, the majority of local sites will use a strict 2PL scheme for local concurrency control.

Consequently, the MDAS coordinates the execution of global transactions without the knowledge of any control information from local DBMS. The only information (loss of local autonomy) required by the algorithm is the type of concurrency control protocol performed at the local sites, i.e., locking, time stamp, unknown, etc.

The semantic information contained in the summary schemas is used to maintain global locking tables. As noted before, each node in the summary schema hierarchy captures the semantic contents of its children in a hierarchical fashion (see Section 2.1.2). As a result, the "data" item being locked is reflected either exactly or as a hypernym term in the summary schema of the transaction's GTM. The locking tables can be used in an aggressive manner where the information is used only to detect potential global deadlocks. A more conservative approach can be used where the operations in a transaction are actually delayed at the GTM until a global lock request is granted. Higher throughput at the expense of lower reliability is the direct consequence of the application of semantic contents rather than exact contents for an aggressive approach. In either case, the global locking table is used to create a global wait-for-graph, which is subsequently used to detect and resolve potential global deadlocks.

Definition 4. *The wait-for-graph is a directed graph that shows dependence/independence among global transactions. Each node in the graph represents a transaction and an edge* $T_i \rightarrow T_j$, *from node* T_i *to node* T_j *represents the fact that transaction* T_i *is waiting for transaction* T_j *to release some lock. In a distributed environment, each scheduler at site i can maintain a local wait-for-graph. In addition, the global manager also maintains a global-wait-for-graph that is the union of all local wait-for-graphs. In the proposed v-locking algorithm, due to the hierarchical nature of the summary schemas model (Fig. 1), the global wait-for-graphs are maintained in hierarchical fashion within the summary schemas nodes—each summary schemas node acts as the global coordinator (see Definition 1) for queries and subqueries under its control and, hence, it maintains a wait-for-graph for them. In addition, edges in the wait-for-graphs, as discussed later are labeled as exact or imprecise.*

The accuracy of the "waiting information" contained in the graph is dependent upon the amount of communication overhead that is required. The proposed algorithm can dynamically adjust the frequency of the communications (acknowledgment signals) between the GTM and local sites based on the network traffic and/or a threshold value. Naturally, the decrease in communication between the local and global systems comes at the expense of an increase in the number of potential false aborts. The extent of detection of false deadlocks is studied in the simulation.

The pseudocode for the global locking algorithm is given in Figs. 2 and 3. Fig. 2 describes how the wait-for-graph is constructed based upon the available communication. Three cases are considered: 1) Each operation in the transaction is individually acknowledged, 2) write operations are only acknowledged, and 3) only the commit or abort of the transaction is acknowledged. For the first case, based upon the semantic contents of the summary schema node, an edge inserted into the wait-for-graph is marked as being an exact or imprecise data item. For each acknowledgment signal received, the corresponding edge in the graph is marked as exact. In the second case, where each write operation generates an acknowledgment signal, for each signal only the edges preceding the last known acknowledgment are

```
Repeat
Case 1, 2, and 3 are chosen based upon the available bandwidth:
Case 1: Acknowledge each operation
        Repeat until commit or abort
        For each operation in a global sub-transaction
                If the lock for the data item is free
                        Acquire lock and add the entry into the lock table
                        Mark query as exact or imprecise
                Else
                        Queue the data item into the locking table
                        Enter an edge based upon semantic information into the wait-for-graph
                        Mark the edge as acknowledged, exact or imprecise
                        Wait for free lock
                Wait for acknowledgement
                Mark the acknowledgement
        End repeat
Case 2: Acknowledge write operations only
        Repeat until commit or abort
        For each operation in a global sub-transaction
                If the lock for the data item is free
                        Acquire lock and add the entry into the lock table
                        Mark query as exact or imprecise
                 Else
                        Queue the data item into the locking table
                        Enter pending edge based upon semantic information into the wait-for-
                        graph and mark it as precise or imprecise
                        Wait for free lock
        For each write operation
                Wait for acknowledgement
                        Mark current and all proceeding read operations as acknowledged
        End repeat
Case 3: Acknowledge commit or abort only
        For each operation in a global sub-transaction
                If the lock for the data item is free
                        Acquire lock and add the entry into the lock table
                        Mark query as exact or imprecise
                Else
                        Queue the data item into the locking table
                        Enter pending edge into wait-for-graph
                        Mark as precise or imprecise
                        Wait for free lock
        Wait for commit or abort
        Mark operations as acknowledged
```

Fig. 2. Global locking algorithm—insertion of edges.

```
Repeat every chktime threshold
        Check for cycles /* depth first search (DFS) is used for cycle detection */
        For each cycle detected
                If all involved in cycle are exact data items AND acknowledged
                        Choose victim and break deadlock /* victim is chosen based upon
                                "progress" */
                /* else if the time threshold for imprecise data is reached
                ElseIf time_elapsed > imprecise_data_time for all transactions in cycle
                        AND All are acknowledged
                        Choose victim and break deadlock
                /* if the ack passes a time threshold */
                Else time_elapsed > acktime for all non acknowledged transactions in
        cycle                                   Choose victim and break deadlock
End repeat
```

Fig. 3. Global locking algorithm—detection of cycles.

marked as being exact. Other edges that have been submitted but that have not been acknowledged are marked as pending. As in the previous two cases, in the third case, the edges are marked as representing exact or imprecise data. However, all edges are marked as pending until the commit or abort signal is received. Keeping the information about the data and status of the acknowledgment signals enables us to detect cycles in the wait-for-graph.

Fig. 3 shows how to detect cycles in the wait-for-graph based on the depth first search (DFS) policy [15]. The graph

is checked for cycles after a time threshold for each transaction. For all of the transactions involved in a cycle, if the exact data items are known and all of the acknowledgments have been received, then a deadlock is precisely detected and broken. When imprecise data items are present within a cycle, the algorithm will consider the cycle a deadlock only after a longer time threshold has passed. Similarly, a pending acknowledgment of a transaction is only used to break a deadlock in a cycle after an even longer time threshold has passed. The time thresholds can be selected and adjusted dynamically to prevent as many false deadlocks as possible.

To clarify the sequence of the operations we will focus on several scenarios:

- Assume a global transaction is resolved in an SSM node (this SSM node is then the GTM of this global transaction) without any conflict with other global transactions resolved in the same SSM node. Further assume that global locks are requested and granted for all the data items acquired by this transaction. Consequently, at this SSM node, the global locking table is updated and a node is inserted in the global wait-for-graph. Under these conditions, GTM will initiate execution of related subtransaction.
- Assume a global transaction is resolved in an SSM node with at least a conflict with another global transaction resolved in the same SSM node. Under this situation, the global locking table is updated and a node representing this transaction, along with dependence link(s), is added to the global wait-for-graph. Note that a dependence link is labeled as exact or imprecise based on the degree of similarity between the query term and entries in summary schema.
- Assume a global transaction completes its operations. This releases some resources; consequently, the global locking table and the global wait-for-graph of corresponding GTM are updated. This could allow some global transaction(s) that are resolved in the same SSM to be ready for execution. Now, an independent global transaction takes steps to request global lock for its data items.

It should be noted that the above sequence of operations would be done recursively within the SSM hierarchy. In another words, insertion of entries in the locking table and wait-for-graph at the GTM node of a global transaction would also initiate update at the lower level summary schemas nodes where the corresponding subtransactions are resolved.

A potential deadlock situation may also occur due to the presence of indirect conflicts. By adding site information to the global locking tables, an implied wait-for-graph could be constructed using a technique similar to the potential conflict graph algorithm [8]. A potential wait-for-graph is a directed graph with transactions as nodes. The edges are inserted between two transactions for each site where there are both active and waiting transactions. The edges are then removed when a transaction aborts or commits. A cycle in the graph indicates the possibility that a deadlock has occurred.

Here, the term active simply means that the transaction has begun execution at a site and is either actively processing or waiting for a blocked resource. For the transactions that are waiting, it is much more difficult to determine exactly which resource is not available. In particular, indirect conflicts, where global transactions are waiting for some local transaction, are not exactly detected. Since the status of the locks at the local sites is not known, there is no way to accurately determine this information without severely violating the autonomy of the local DBMS. Therefore, the potential wait-for-graph is used to detect potential deadlocks. The actual deadlocks in the system are a subset of the deadlocks that are contained in the implied wait-for-graph. Thus, as is the case when detecting deadlocks using global locking, there is also the potential for false deadlock detection. To decrease the number of false deadlocks, the potential wait-for-graph is used in conjunction with a waiting period threshold. The waiting period threshold is longer than the maximum time threshold used in the global locking tables. This allows the global locking algorithm to "clear" as many deadlocks as possible and, hence, reduces the possibility of detecting false cycles in the potential wait-for-graph.

3.1.1 Handling Unknown Local Data Sources

Our algorithm was extended to handle the local "black box" site in which the global level knows nothing about the local concurrency control. Since nearly every commercial database system uses some form of 2PL, this case will only comprise a small percentage of local systems. The algorithm merely executes global transactions at such a site in a serial order. This is done by requiring any transaction involving the "black box" to obtain a site lock before executing any operations in the transaction. These types of locks will be managed by escalating any lock request to these sites to the highest level (site lock).

3.2 Proposed Data Replication Protocol

3.2.1 Communication Protocol

Maintaining the ACID properties of a transaction with replicated data in an MDAS environment is very difficult. The proposed scheme considers three levels of connectivity in which a mobile unit operates. During a *strong connection*, the mobile unit sends/receives all transactions and returns data directly to/from land-based, fixed sites for processing. When the communication link degrades to a *weak connection*, transactions are queued at the mobile unit and passed through the system according to the availability of bandwidth. Returned data is also queued at the fixed site. The queuing of operations during a weak connection allows a mobile unit to continue processing at the expense of increased latency.

In the disconnected state, a user may perform read-only queries on any cached data. For this case, the consistency of the data is not guaranteed, i.e., the user may receive stale data. Any transaction that contains a write operation is queued and submitted to the system when the connection is reestablished. Naturally, if a read-only access does not find the data locally, the query is queued and submitted later when the connection is established.

3.2.2 Cached Data—Bundled Queries (BUNQ)

Data that is cached on the mobile unit consists of a query and its associated data. There are several reasons for using a BUNQ instead of page-based or file-based data. In a tightly

coupled distributed system, it is possible to cache data at the local unit. However, in a multidatabase system, the structure of the data at the local databases will vary (*structural differences*—the information may be stored as structured data, unstructured data, web pages, files, objects, etc.). This variaition of data at each local site makes it difficult to cache the data in a uniform manner. In addition, the *autonomy requirement* of the local database imposes further restrictions for caching data. It may not be possible to determine the underlying structure of the data at the local site without violating local autonomy requirements. For instance, to determine the pages of data accessed in a query from a local DBMS, knowledge regarding the tuples as well as the mapping to pages may be required. In order to provide this information, the local DBMS would have to be modified, directly violating autonomy requirements. Furthermore, the invalidation of the data at the local sites should be propagated to the mobile unit. This would also require major changes to the local systems and, hence, would violate local autonomy.

Consequently, instead of caching individual data items from each local source, the data set associated with a particular transaction is cached—a bundled query. By caching a BUNQ, the resolution of the structural differences is done at the MDAS level while still maintaining the local autonomy requirements. The primary advantage is the ease and simplicity of implementation. This comes at the expense of retaining data in the cache at a very coarse grained level.

3.2.3 Prefetching and Replacement/Invalidation Policy

- **Prefetching**. The proposed scheme uses a simple prefetching and replacement policy. The literature has addressed prefetching algorithms based upon user profiles and usage histories [30], [34], [40]. The limited power, storage, processing capability, and bandwidth of a mobile unit make the incorrect prefetch of data extremely expensive. The idea is to prefetch enough data such that the user can still operate during a disconnection (albeit with relaxed consistency requirements) while minimizing the use of additional energy and bandwidth.

  Allowing the user to specify a particular read-only transaction as an automated queued query prefetches the data. An AQ^2 has a relaxed requirement for consistency, which is defined by the user. The user sets a valid time threshold for each AQ^2 when defining such a transaction. The mobile unit automatically submits the transaction to the MDAS when the threshold has expired—i.e., prefetches the data. The results are stored as a BUNQ. If the user requests the data in the AQ^2 before the BUNQ is invalidated, the query is serviced from the local cache.
- **Replacement**. The data in the cache consists of both automated queued queries and other user submitted read-only queries. The data in the cache is replaced based upon the least recently used (LRU) policy. The LRU policy has its advantages in that it is well-understood and easy to implement. Moreover, other than some web-based caching algorithms, the LRU policy is the most widely used replacement policy in DBMS caches [12], [24].
- **Invalidation**. To maintain consistency between the copies of data residing on the fixed and mobile units, the data in the cache must be correctly invalidated when the main copy changes. To accomplish this, we propose using a parity-based signature for each BUNQ in the cache. When the user submits a transaction, if a corresponding BUNQ is present in the cache, the transaction (along with the parity code) is sent to the fixed node. The fixed node then performs the query and delays the transmission of the information back to the mobile unit until it generates and compares the two parity codes. If they are identical, only an acknowledgment is sent back to the mobile unit and the data is read locally from the cache. Otherwise, the resultant data, along with its new parity sequence, is returned and replaced in the cache. The old copy of the BUNQ is invalidated according to the LRU rules.

The use of a parity signature for invalidation incurs a small overhead. The parity sequence is generated for every user submitted read-only transaction, and once for each time quantum of the AQ^2. The only additional overhead that the mobile unit incurs is the added space required for storing the parity sequence. The fixed systems (assumed to have a relatively high degree of processing power) will perform all of the parity generation and validation. A high degree of certainty is assured by using a relatively small number of bits and standard error detection or correction techniques [37]. The number of bits required should be less than 64 bits per BUNQ. With 64 bits of parity, the approximate probability of incorrectly detecting a matching BUNQ is $1/2^{64}$.

4 Evaluation of Proposed Algorithm

4.1 Simulation

The performance of the proposed v-locking and p-caching algorithms was evaluated through a simulator written in C++ using CSIM. The simulator measures performance in terms of global transaction throughput, response time, and CPU, disk I/O, and network utilization. In addition, the simulator was extended to compare and contrast the behavior of the v-lock algorithm against the site-graph, potential conflict graph, and the forced conflict algorithms. This includes an evaluation with and without the cache.

The MDAS consists of both local and global components, as well as mobile units. The local component is comprised of local database systems, performing local transactions outside the control of the MDAS. The global component consists of transactions executing under the control of the MDAS. The mobile units generate the actual global transactions. Figs. 4, 5, and 6 depict the flow of operations in the mobile, global, and local units, respectively.

There are a fixed number of active global transactions present in the system at any given time. An active transaction is defined as being in the active, CPU, I/O, communication, or restart queue. A global transaction received from a mobile unit enters a communication queue, and subsequently enters the active queue. Each operation of the transaction is scheduled and is communicated to the

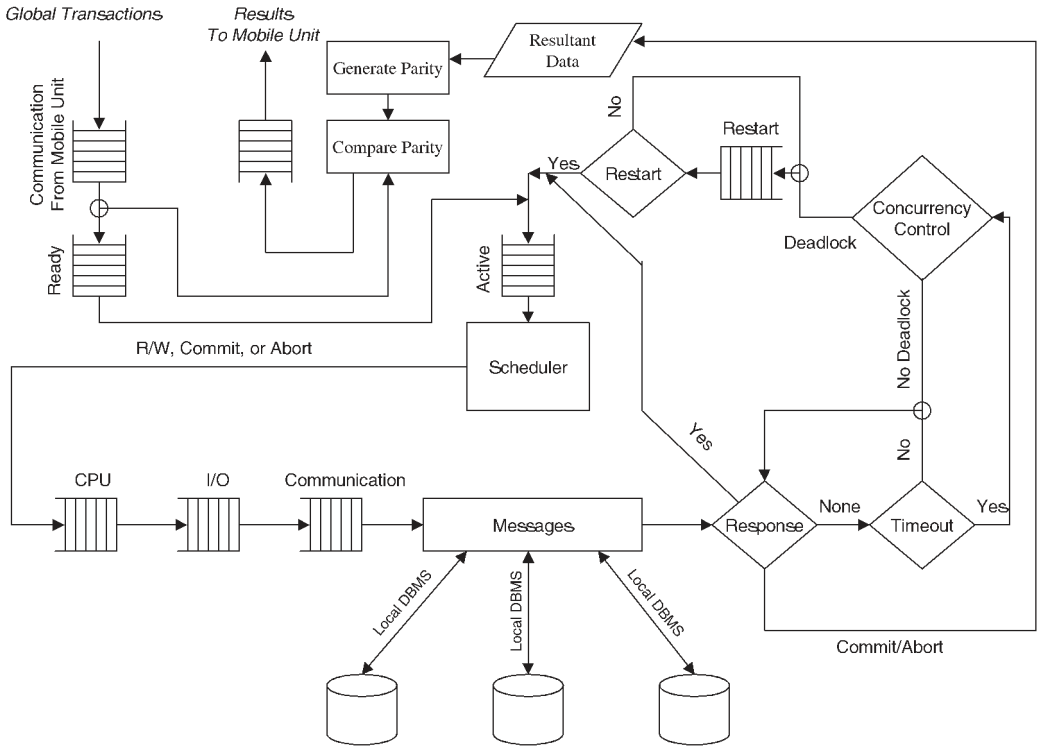


Fig. 4. Flow of operations in the mobile unit.

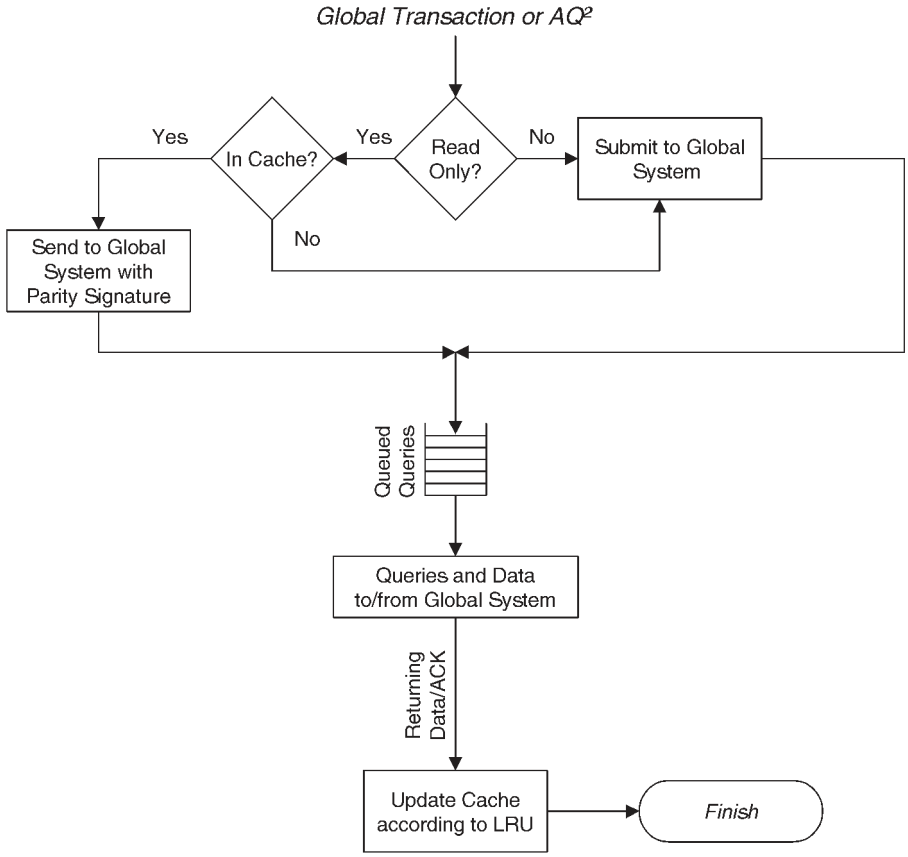


Fig. 5. Global system architecture.

local system based upon the available bandwidth. The global scheduler acquires the necessary global virtual locks and processes the operation. The operation(s) then uses the CPU and I/O resources and is communicated to the local system based upon the available bandwidth. When acknowledgments or commit/abort signals are received

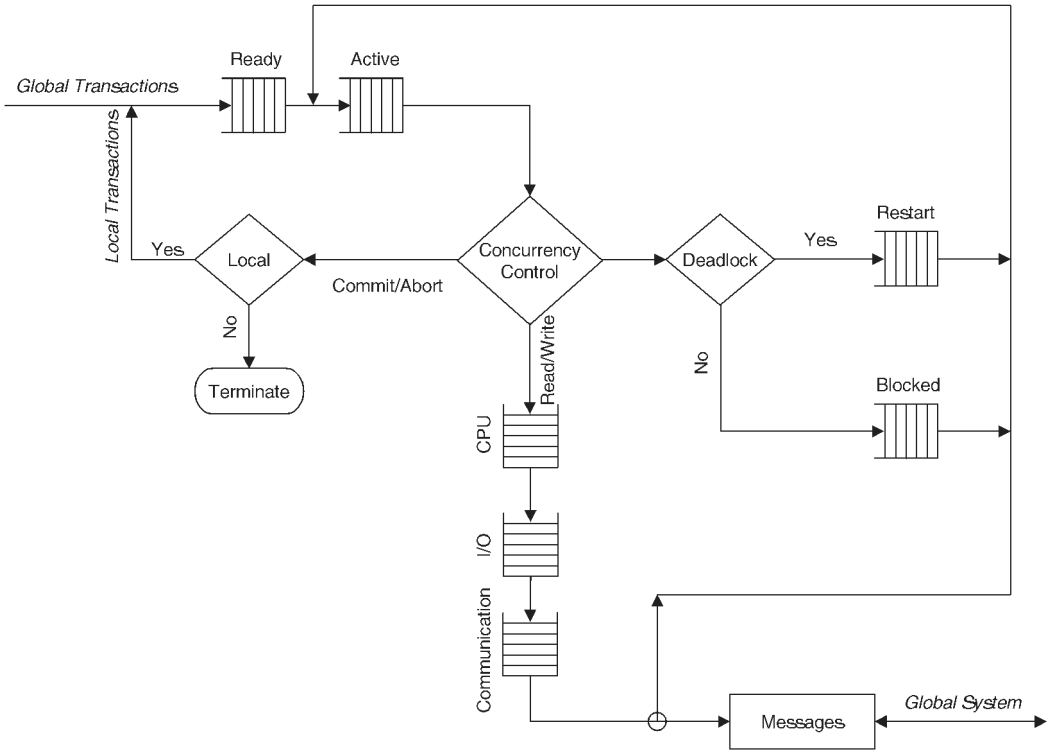


Fig. 6. Local system architecture.

from the local site, the algorithm determines if the transaction should proceed, commit, or abort. For read-only transactions following a global commit, a parity code is generated for the resultant data and compared to the parity code of the BUNQ. For a matching code, only an acknowledgment signal is sent back to the mobile unit. Otherwise, the data and the new parity code are sent back to the mobile unit. Transactions containing a write operation are placed directly in the ready queue. If a deadlock is detected, or an abort message is received from a local site, the transaction is aborted at all sites and the global transaction is placed in the restart queue. After a specified time elapses, the transaction is again placed on the active queue.

At the local sites, there are a fixed number of active local transactions. The active transactions are comprised of both local transactions and global subtransactions. The local system does not differentiate between the two types. An active transaction is defined as being in the active, CPU, I/O, communication, blocked, or restart queue. Transactions enter the active queue and are subsequently scheduled by acquiring the necessary lock on a data item. If the lock is granted, the operation proceeds through the CPU and I/O queue and, for global subtransactions, an acknowledgment signal is communicated back to the global locking scheme based upon the available communication bandwidth. If a lock is not granted, the system checks for deadlocks and will place the transaction either in the blocked queue, or the restart queue. For local transactions, it goes into the restart queue if it is aborted and, subsequently, it will be restarted later. Upon a commit, a new local transaction is generated and placed in the ready queue. For global subtransactions, an abort or commit signal is communicated back to the GLS and the subtransaction terminates.

4.2 System Parameters

The underlying global information sharing process is composed of 10 local sites. The size of the local databases at each site can be varied and has a direct effect on the overall performance of the system. The simulation is run for 5,000 time units and an average of 10 runs is taken for the values presented. The global workload consists of randomly generated global queries, spanning over a random number of sites. Each operation of a subtransaction (read, write, commit, or abort) may require data and/or acknowledgments to be sent from the local DBMS. The frequency of messages depends upon the quality of the network link. In order to determine the effectiveness of the proposed algorithm, several parameters are varied for different simulation runs. These parameters for the global system are given in Table 3, along with their default values.

A collection of mobile units submits global queries (selected from a pool of 500 queries) to the MDAS. The connection between the mobile unit and the MDAS has a 50 percent probability of having a strong connection and a 30 percent probability of having a weak connection. There is a 20 percent probability of being disconnected. A strong connection has a communication service time of 0.1 to 0.3 seconds, while a weak connection has a service time range of 0.3 to 3 seconds. When a disconnection occurs, the mobile unit is disconnected for 30 to 120 seconds. Initially, one-third of the pool consists of read-only queries, which are locally cached as a BUNQ. Additionally, 10 percent of the read-only queries are designated as an AQ^2. For read-only queries, the local cache is first checked for an existing BUNQ. If present, the query is submitted along with the associated parity sequence. The MDAS returns either the data or an acknowledgment signal for a matching BUNQ. Subsequently, if signatures do not match, the mobile unit updates

TABLE 3
Global Parameters

Global System Parameters	Default Value
The number of local sites in the system	10
The number of data items per local site	100
The maximum number of global transactions in the system. This number represents the global multiprogramming level.	10
The maximum number of operations that a global transaction contains.	8
The minimum number of operations that a global transaction contains.	1
The service time for the CPU queue	.005 sec
The service time for the IO queue	.010 sec
The service time for each communicated message to the local site	.100 sec
The number of messages per operation (read/write)	2

TABLE 4
Mobile Unit Parameters

Mobile Unit Parameters	Default Value
The maximum number of mobile units in the system. This number represents the global multiprogramming level.	10
The maximum number of operations that a global transaction contains.	8
The minimum number of operations that a global transaction contains.	1
The service time for the CPU queue	0.005 sec
The service time for the IO queue	0.010 sec
The service time for each communicated message to the global system for a strong connection, randomly selected	0.100-0.300 sec
The service time for each communicated message to the global system for a weak connection, randomly selected	0.300-3 sec
The service time for a disconnection, randomly selected	30-120 sec
The probability of a strong connection	0.50
The probability of a weak connection	0.30
The probability of a disconnection	0.20
The number of messages per operation (read/write)	2
The average size of each message	1,024 bytes
The size of the mobile unit cache	1 Megabyte
The probability that a query is read-only	1/3
The probability that a read-only query is submitted as an AQ^2	0.1

the cache with the new data according to the LRU scheme. Upon termination of a transaction, a new query is selected and submitted to the MDAS. The various parameters for the mobile units are given in Table 4, along with their default values.

The local systems perform two types of transactions—local and global. Global subtransactions are submitted to the local DBMS and appeared as local transactions. Local transactions generated at the local sites consist of a random number of read/write operations. The only difference between the two transactions is that a global subtransaction will communicate with the global system, whereas the local transaction terminates upon a commit or abort. The local system may abort a transaction, global or local, at any time. If a global subtransaction is aborted locally, it is communicated to the global system and the global transaction is aborted at all sites. The various parameters for the local system are given in Table 5 along with their default values. Both the global and local systems are modeled similar to the models used in [2], [8], [31].

4.3 Simulation Results

The simulator compared and contrasted the v-lock, potential conflict graph (PCG), forced-conflict, and site-graph algorithms with and without the proposed p-caching scheme (Figs. 7 and 8). As can be concluded, the v-Lock algorithm has the highest throughput. This result is consistent with the fact that the v-Lock algorithm is better able to detect global conflicts and thus achieves higher concurrency than the other algorithms. As can be seen, the maximum throughput occurs at a multiprogramming level approximately equal to 40. As expected, as the number of concurrent global transactions increases, the number of completed global transactions decreases due to the increase in the number of conflicts. The peak throughput for each concurrency control scheme is slightly higher with the use of caching. Furthermore, the throughput for each number of active global transactions is also higher than the noncaching case. This is attributed to the cache hits on the read only data. For the noncache case, the throughput is low until the active number of global transactions reaches about 30, with a rapid increase of the throughput from 30 to 40 active transactions. This occurs because of the weak connections

TABLE 5
Local Parameters

Global System Parameters	Default Value
The maximum number of local transactions per site. This number represents the local multiprogramming level.	10
The maximum number of write operations per transaction	8
The maximum number of read operations per transaction	8
The minimum number of write operations per transaction	1
The minimum number of read operations per transaction	1
The service time for the local CPU queue	.005
The service time for the local IO queue	.010
The service time for each communicated message to the MDAS	.100
The number of messages per operation (read/write)	2

and disconnection. With the p-caching algorithm, the rate of the increase in throughput is more gradual because the local cache can service the read-only queries under weak connectivity or disconnection.

The gain in the throughput—sensitivity—of the p-caching algorithm upon the different concurrency control schemes is shown in Fig. 9. The v-locking scheme shows the greatest sensitivity to the caching algorithm. At 20 active global transactions, there is an improvement in the throughput of approximately 0.6 when using a cache. At the peak throughput of 40 simultaneous transactions, the throughput is increased by 0.2. The PCG site-graph, and forced conflict algorithms show similar characteristics to the v-locking algorithm; however, the sensitivity of these algorithms is less. The caching becomes ineffective for all schemes when the number of active global transactions is greater than 75.

Figs. 10 and 11 show the percentage of completed transactions that the simulator measured for the various concurrency control algorithms. In general, for all schemes, the number of completed transactions decreases as the number of concurrent transactions increases due to more conflicts among the transactions. However, the performance of both the forced conflict and site-graph algorithms decreases at a faster rate. This is due to the increase in the number of false aborts detected by these algorithms. The v-Lock more accurately detects deadlocks by differentiating between global and indirect conflicts, and therefore performs better than the PCG algorithm. At levels below the peak throughput of 40 active transactions, all of the concurrency control schemes using the proposed caching algorithm perform better than the noncaching case. With caching, the range of completed transactions through 30 active users for all schemes is 70 to 90 percent. However, without caching, the range falls substantially to 27-78 percent. The increase in completed transactions is due to the cache servicing read-only transactions as well as the queuing of transactions during disconnection. Particularly for the nonlock based schemes—site-graph and forced-conflict—the improvement in the completed transactions is significant.

The communication utilization was found to decrease with the use of the cache. Figs. 12 and 13 show the results of the simulation. Lower than peak throughput, the amount of communication utilization is lower with the use of caching. At 20 active global transactions, the v-locking algorithm utilizes 40 percent of the communication bandwidth versus 69 percent utilization without the cache. Similarly, with 30 active global transactions, there is a 75 percent versus 91 percent communication utilization with and without the cache, respectively. This is attributed to the reduction in transferred data from parity acknowledgments and local accesses to the cache. At peak throughput, both locking algorithms (v-locking and PCG) are utilizing near 100 percent of the communication channel. The communication utilization is near 100 percent at peak throughput and decreases slightly as the number of concurrent transactions increases. It is easy to determine from this graph that the communication requirements for the v-locking algorithm represent the bottleneck of the system for both the caching and noncaching case.

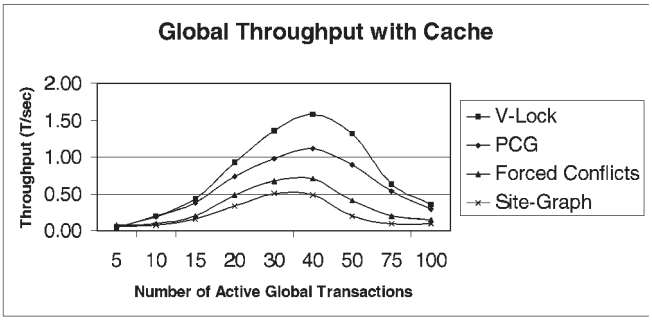


Fig. 7. Global throughput with P-caching.

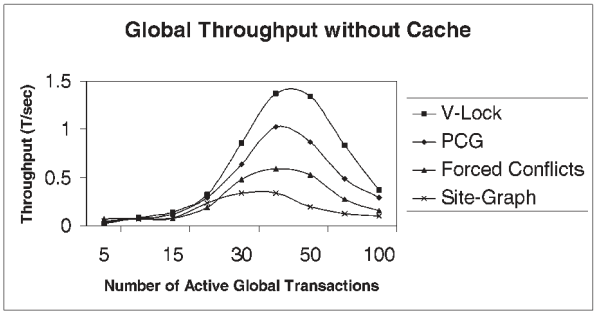


Fig. 8. Global throughput without P-caching.

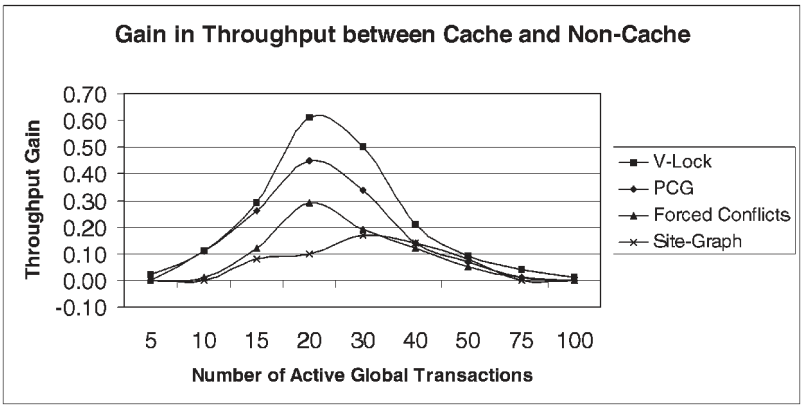


Fig. 9. Comparison of the sensitivity of the P-caching algorithm.

5 Conclusions and Future Directions

5.1 Conclusion

The requirements of an "any time, anywhere" computing environment motivate new concepts that effectively allow a user to access information in a timely and reliable manner. An overview of the characteristics of a new computing environment, the MDAS, and the requirements and issues of this environment were introduced and discussed.

In an MDAS, a potentially large number of users may simultaneously access a rapidly increasing amount of aggregate, distributed data. In such an environment, a concurrency control algorithm must offer higher throughput in the face of the limitations imposed by technology, reduce communications, while preserving as much local autonomy as possible, as well as minimizing communication overhead in the system.

A new, distributed, hierarchically organized concurrency control algorithm was presented and evaluated. The semantic information contained within the SSM is used to maintain the global locking tables in the v-locking algorithm. The global locking tables are used to serialize conflicting operations of global transactions and detect and break deadlocked transactions. This algorithm increases the global performance by dynamically adjusting the amount of communication required to detect and resolve conflicts.

Data duplication in the form of replication and caches was also used to lessen the effects of weak communications or disconnection. The duplicated data at the mobile node allows the user to continue to work in case of a weak connection or disconnection. Automated queued queries and bundled queries were used to address the limited bandwidth and local autonomy restrictions in an MDAS environment. To maintain data consistency between fixed and mobile units, read-only queries are cached as a bundled query and are validated using a simple parity-checking scheme. Automated queued queries are used in order to prefetch data to the mobile unit.

A simulator was developed in C++ using CSIM to evaluate the performance of these algorithms. The simulator compared and contrasted the proposed V-locking algorithm to the PCG, site graph, and forced conflict algorithms. The increased throughput, as well as a significant increase in the percentage of completed transactions, showed that all of the algorithms benefit from the use of the cache. The communication requirements were reduced in all cases with the use of the cache. Furthermore, it was shown how the proposed algorithm decreases the communication requirements, resulting in higher global performance. The performance of the v-locking algorithm was determined to be better than the other schemes in throughput, response time, and resource utilization. The results also verify that communication between the local and global systems is the bottleneck and, thus, the limiting factor in throughput and response time of the global system.

5.2 Future Directions

Although the results we have demonstrated are very promising, the work presented in this paper can be extended in several directions:

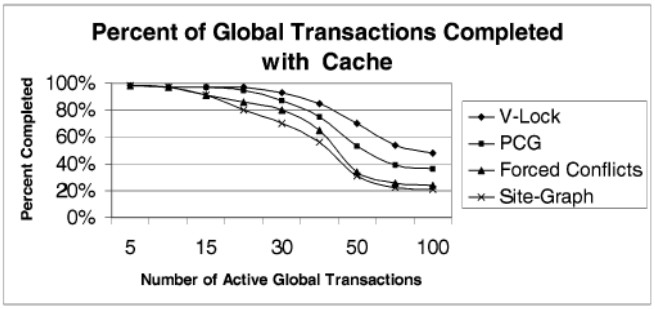


Fig. 10. Comparison of the global transactions completed with cache.

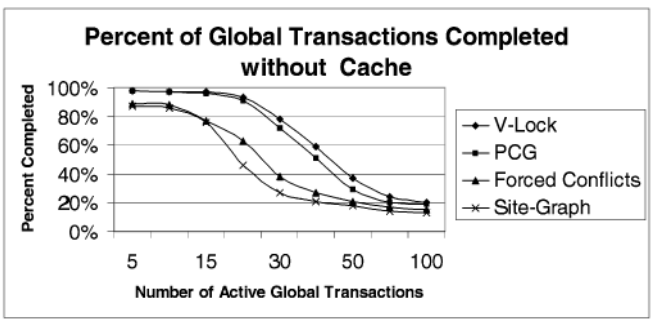


Fig. 11. Comparison of the global transactions completed without cache.

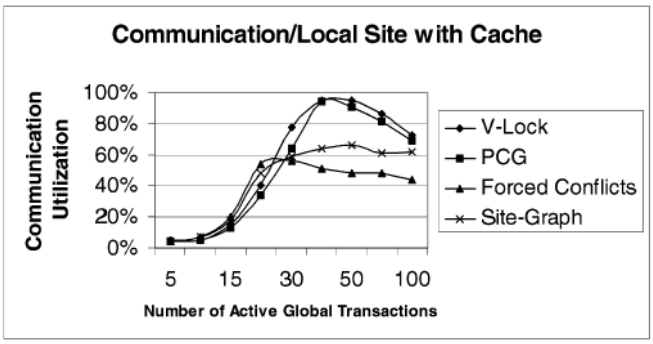


Fig. 12. Communication utilization at each local site with cache.

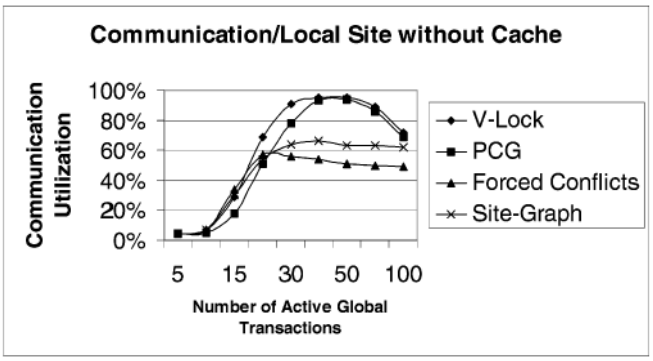


Fig. 13. Communication utilization at each local site without cache.

- The effect of changing the ratio of global to local transactions active in the system should be investigated. This is particularly important for systems that have a very high global transaction requirement. If the global subtransactions were allowed to dominate the local systems, the overall global throughput would increase, while the throughput of local systems may decrease. It should be determined if an optimal ratio or even a range of the ratio can be found.
- As the architecture of the system is hierarchical, there may be several global coordinators active at any time. The current algorithm assumes that any conflicts between these coordinators are not communicated directly between the coordinators, but manifest themselves in the form of an indirect conflict. If the coordinators were allowed to communicate information, would there be a significant impact on the global performance?
- The effect of various parameters upon the number of completed/aborted transactions is being investigated. An extensive study on the effects of changing the number of sites, distribution of data, processing, I/O, and communication is needed. In addition, the impact of nonuniform communication requirements between the client and server as well as between servers should be investigated.
- The impact of the Internet on data management is growing at a tremendous pace. Consideration of how this data (both structured and unstructured) should be integrated into MDAS systems is important.
- The data set for the AQ^2 is assumed to remain static for a minimum of at least two accesses and a maximum of 10 accesses. This is done in order to simulate slower changing data and to see the effectiveness of the cache size. The simulated workload is not indicative of the possible workloads in an MDAS. The effect of different workloads and data sets should be investigated.
- The effect of the cache hit ratio on the MDAS environment (throughput, utilization, etc.) should be studied. This could be done by changing the cache-hit ratio to a raw probability that a read-only query is valid or invalid. The effect of the cache size could also be studied under these conditions. By setting the cache hit ratio instead of relying upon the actual invalidation of the data, one should roughly be able to simulate different workloads and, subsequently, see the effects on the various parts of the MDAS.
- The proposed v-locking and p-caching algorithms showed promising results in the simulation. However, the algorithms should be tested on real database systems.
- Finally, the stationary nodes in the system are assumed to store most of the data in the MDAS. The case in which the data in the system primarily resides on mobile units should be considered.

ACKNOWLEDGMENTS

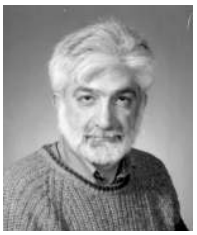
This work in part has been supported by the US Office of Naval Research under the contract N00014-02-1-0282.

REFERENCES

[1] A. Adya and B. Liskov, "Lazy Consistency Using Loosely Synchronized Clocks," *Proc. ACM Symp. Principles of Distributed Computing,* Aug. 1997.
[2] R. Agrawal, M. Carey, and M. Livny, "Models for Studying Concurrency Control Performance: Alternatives and Implications," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 1985.
[3] B. Badrinath, "Designing Distributed Algorithms for Mobile Computing Networks," *Computer Comm.,* vol. 19, no. 4, Apr. 1996.
[4] B. Badrinath and T. Imielinski, "Replication and Mobility," *Proc. Second IEEE Workshop Management of Replicated Data,* Nov. 1992.
[5] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.
[6] Y. Breitbart, A. Silberschatz, and G. Thompson, "An Update Mechanism for Multidatabase Systems," *IEEE Data Eng. Bull.,* vol. 10, no. 3, pp. 12-18, Sept. 1987.
[7] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz, "Overview of Multidatabase Transaction Management," *Very Large Databases J.,* vol. 1, no. 2, pp. 181-239, Oct. 1992.
[8] Y. Breitbart and A. Silberschatz, "Performance Evaluation of Two Multidatabase Transaction Management Algorithms," *Computing Systems,* vol. 6, no. 3, Summer 1993.
[9] M. Bright, A. Hurson, and S. Pakzad, "A Taxonomy and Current Issues in Multidatabase Systems," *Computer,* vol. 25, no. 3, pp. 50-60, Mar. 1992.
[10] M.W. Bright, A.R. Hurson, and S.H. Pakzad, "Automated Resolution of Semantic Heterogeneity in Multidatabases" *ACM Trans. Database Systems,* vol. 19, no. 2, 1994.
[11] O. Bukhres, "Performance Comparison of Distributed Deadlock Detection Algorithms," *Proc. Int'l Conf. Data Eng.,* 1992.
[12] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World Wide Web," *IEEE Trans. Computers,* vol. 47, no. 4, Apr. 1998.
[13] Y.C. Chehadeh, A.R. Hurson, and D. Tavangarian, "Object Organization on Single and Parallel Broadcast Channel," *High Performance Computing,* 2001.

[14] P. Chrysanthis, "Transaction Processing in Mobile Computing Environment," *Proc. IEEE Workshop Advances in Parallel and Distributed Systems,* Oct. 1993.
[15] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms.* McGraw-Hill, 1990.
[16] K. Dash, A. Hurson, S. Phoha, and C. Chehadeh, "Summary Schemas Model: A Scheme for Handling Global Information Sharing," *Proc. Int'l Conf. Intelligent Information Management Systems,* pp. 47-51, 1994.
[17] A. Demers, K. Pertersen, M. Spreitzer, D. Terry, M. Theier, and B. Welch, "The Bayou Architecture: Support for Data Sharing among Mobile Users," *IEEE Proc. Workshop Mobile Computing Systems and Applications,* Dec. 1994.
[18] T. Devirmis and O. Ulosoy, "Design and Evaluation of a New Transaction Execution Model for Multidatabase Systems," *Information Sciences,* vol. 102, pp. 203-238, 1997.
[19] W. Du and A. Elmagarmid, "Quasi Serializability: A Correctness Criterion for Global Concurrency Control in InterBase," *Proc. 15th Int'l Conf. Very Large Databases,* pp. 347-355, 1992.
[20] M.H. Dunham, A. Helal, and S. Balakrishnan, "A Mobile Transaction Model that Captures Both the Data and Movement Behavior," *Mobile Network Applications,* pp. 149-162, Oct. 1997.
[21] A. Elmagarmid, J. Jing, and T. Furukawa, "Wireless Client/Server Computing for Personal Information Services and Applications," *ACM Sigmod Record,* 1995.
[22] M. Faiz, A. Zaslavsky, and B. Srinivasan, "Revising Replication Strategies for Mobile Computing Environments," *Proc. ECOOP 95 Workshop Mobility and Replication,* 1995.
[23] A.A. Farrag and M.T. Ozsu, "Using Semantic Knowledge of Transactions to Increase Concurrency," *ACM Trans. Database Systems,* vol. 14, no. 4, pp. 503-525, Dec. 1989.
[24] M. Franklin, M. Carey, and M. Livny, "Transactional Client-Server Cache Consistency: Alternatives and Performance," *ACM Trans. Database Systems,* vol. 22, no. 3, Sept. 1997.
[25] D. Geogakopoulos, M. Rusinkiewiez, and A. Sheth, "Using Tickets to Enforce the Serializability of Multidatabase Transactions," *IEEE Trans. Knowledge and Data Eng.,* vol. 6, no. 1, pp. 166-180, Feb. 1994.
[26] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.
[27] J. Gwertzman and M. Seltzer, "World-Wide Web Cache Consistency," *Proc. 1996 USENIX Technical Conf.,* Jan. 1996.
[28] P. Honeyman, L. Huston, J. Rees, and D. Bachmann, "The LITTLE WORK Project," *Proc. Third IEEE Workshop Workstation Operating Systems,* Apr. 1992.
[29] D. Kottmann, "Serializing Operations into the Past and Future: A Paradigm for Disconnected Operations on Replicated Objects," *Proc. ECOOP 95 Workshop Mobility and Replication,* 1995.
[30] H. Lei and D. Duchamp, "An Analytical Approach to File Prefetching," *Proc. 1997 USENIX Ann. Technical Conf.,* Jan. 1997.
[31] J. Lim and A. Hurson, "Heterogeneous Data Access in a Mobile Environment—Issues and Solutions," *Advances in Computers,* vol. 49, pp. 119-178, 1999.
[32] S. Mehrotra, H. Korth, and A. Silberschatz, "Concurrency Control in Hierarchical Multidatabase Systems," *The Very Large Database J.,* vol. 6, pp. 152-172, 1997.
[33] J. Metzner, "Efficient Replicated Remote File Comparison," *IEEE Trans. Computers,* vol. 40, no. 5, May 1991.
[34] R.H. Patterson et al., "Informed Prefetching and Caching," *Proc. 15th Symp. Operating System Principals,* Dec. 1995.
[35] E. Pitoura, "A Replication Schema to Support Weak Connectivity in Mobile Information Systems," *Proc. Seventh Int'l Conf. Database and Expert Systems Applications (DEXA),* Sept. 1996.
[36] E. Pitoura and B. Bhargava, "A Framework for Providing Consistent and Recoverable Agent-Based Access to Heterogeneous Mobile Databases," *Sigmod Record,* vol. 24, no. 3, Sept. 1995.
[37] E. Pitoura and B. Bhargava, "Maintaining Consistency of Data in Mobile Distributed Environments," *Proc. 15th Int'l Conf. Distributed Computing Systems,* pp. 404-413, 1995.
[38] C. Pu, G. Kaiser, and N. Hutchinson, "Split-Transactions for Open-Ended Activities," *Proc. 14th Very Large Database Conf.,* 1998.
[39] P. Reiher, J. Heidemann, D. Ratner, G. Skinner, and G. Popek, "Resolving File Conflicts in the Ficus File System," *Proc. 1994 USENIX Conf.,* 1994.
[40] M. Satyanarayanan, "Fundamental Challenges in Mobile Computing," *Proc. 15th ACM Symp. Principles of Distributed Computing,* May 1996.
[41] M. Satyanarayanan, B. Noble, P. Kumar, and M. Price, "Application-Aware Adaptation for Mobile Computing," *Proc. Sixth ACM SIGOPS European Workshop,* Sept. 1994.
[42] M. Satyanarayanan, J.J. Kistler, and L.B. Mummert, "Experience with Disconnected Operation in a Mobile Computing Environment," *Proc. 1993 USENIX Symp. Mobile and Location-Independent Computing,* Aug. 1993.
[43] K. Segun, A.R. Hurson, V. Desai, A. Spink, and L.L. Miller, "Transaction Management in a Mobile Data Access System," *Ann. Rev. Scalable Computing,* vol. 3, 2001.
[44] A. Wolski and J. Veijalainen, "2PC Agent Method: Achieving Serializability in Presence of Failures in a Heterogeneous Multidatabase," *Proc. Int'l Conf. Databases, Parallel Architectures, and Their Applications,* pp. 321-330, 1990.
[45] A. Zhang, M. Bhargava, and O. Bukhres, "Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems," *Proc. ACM-SIGMOD Int'l Conf. Management of Data,* 1994.

James B. Lim received the BSE degree in electrical engineering from the University of Pennsylvania in 1991, the MS degree in 1996, and the PhD degree in 1998, from Pennsylvania State University in computer engineering. His research at Penn State included transaction management in a mobile, distributed environment, more specifically, the research, simulation, and implementation of algorithms that address transaction management issues in a mobile, multidatabase environment. After graduation, he joined the EMC Software Development Center. As a senior principal researcher at EMC, he was responsible for research and development of storage-centric clustering, disaster recovery, and online backup solutions. Currently, he is the chief technical officer at MJL Technology. Current research includes ubiquitous messaging (video, voice, mail, etc.) from any location or device, embedded systems, high-speed optical networks, and 3G/4G wireless systems.



A.R. Hurson is with the Computer Science and Engineering Faculty at Pennsylvania State University. His research for the past 18 years has been directed toward the design and analysis of general as well as special purpose computer architectures. His research has been supported by the NSF, NCR Corp., DARPA, IBM, Lockheed Martin, and Penn State University. He has published more than 200 technical papers in areas including database systems, multidatabases, object-oriented databases, computer architecture and cache memory, parallel and distributed processing, dataflow architectures, and VLSI algorithms. Dr. Hurson served as the guest coeditor of special issues of the IEEE Proceedings on supercomputing technology, the *Journal of Parallel and Distributed Computing* on load balancing and scheduling, the *Journal of Integrated Computer-Aided Engineering on Multidatabase and Interoperable Systems*, *IEEE Transactions on Computers* on parallel architectures and compilation techniques, and the *Journal of Multimedia Tools and Applications*. He is the coauthor of the *IEEE Tutorials on Parallel Architectures for Database Systems, Multidatabase Systems: An Advanced Solution for Global Information Sharing*, *Parallel Architectures for Data/Knowledge Base Systems, and Scheduling and Load Balancing in Parallel and Distributed Systems*. He is also the cofounder of the IEEE Symposium on Parallel and Distributed Processing (currently IPDPS). Dr. Hurson has been active in various IEEE/ACM conferences and has given tutorials for various conferences on global information sharing, dataflow processing, database management systems, supercomputer technology, data/knowledge-based systems, scheduling and load balancing, and parallel computing. He served as a member of the IEEE Computer Society Press Editorial Board and an IEEE Distinguished Speaker. Currently, he is serving on the IEEE/ACM Computer Sciences Accreditation Board, as an editor of the *IEEE Transactions on Computers*, and as an ACM lecturer.