nication services. *Pattern Languages of Program Design* (eds R. Martin, F. Buschmann, and D. Riehle), Reading, MA: Addison-Wesley.
[61] Schmidt, D. C. (1998). Evaluating architectures for multi-threaded CORBA object request brokers. *Communications of the ACM Special Issue on CORBA*, **41**, October.
[62] Kuhns, F., Schmidt, D. C., and Levine, D. L. (1999). The design and performance of a Real-Time I/O subsystem, *Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium*, IEEE, Vancouver, British Columbia, Canada, June.
[63] Schmidt, D. C., and Cleeland, C. (1999). Applying patterns to develop extensible ORB middleware. *IEEE Communications Magazine*, **37**, no. (4), April.
[64] Schmidt, D. C., Gokhale, A., Harrison, T., and Parulkar, G. (1997). A high-performance endsystem architecture for real-time CORBA. *IEEE Communications Magazine*, **14**, February.
[65] Schmidt, D. C., Harrison, T. H., and Al-Shaer, E. (1995). Object-oriented components for high-speed network programming. *Proceedings of the 1st Conference on Object-Oriented Technologies and Systems*, Monterey, CA, USENIX, June.
[66] Schmidt, D. C., Harrison, T., and Pryce, N. (1997). Thread-specific storage—An object behavioral pattern for accessing per-thread state efficiently. C++ report, **9** no. (10), November/December.
[67] Schmidt, D. C., Levine, D. L., and Mungee, S. (1998). The design and performance of real-time object request brokers. *Computer Communications*, **21**, 294–324, April.
[68] Schmidt, D. C., Mungee, S., Flores-Gaitan, S., and Gokhale, A. (1999). Software architectures for reducing priority inversion and non-determinism in Real-time object request brokers, *Journal of Real-time systems*, Kluwer Academic Publishers, to appear.
[69] Schmidt, D. C., and Suda, T. (1994). An object-oriented framework for dynamically configuring extensible distributed communication systems. *IEE/BCS Distributed Systems Engineering Journal (Special Issue on Configurable Distributed Systems)*, **2**, 280–293, December.
[70] Schmidt, D. C., and Vinoski, S. (1997). Object adapters: concepts and terminology. *C++ Report*, **9**, November/December.
[71] Schmidt, D. C., and Vinoski, S. (1998). Using the portable object adapter for transient and persistent CORBA objects. *C++ Report*, **10**, April.
[72] Stankovic, J. A. (1988). Misconceptions about real-time computing. *IEEE Computer*, **21**, 10–19, October.
[73] Stewart, D. B., and Khosla, P. K. (1992). Real-time scheduling of sensor-based control systems. *Real-Time Programming* (eds W. Halang and K. Ramamritham), Tarrytown, NY: Pergamon Press.
[74] Tennenhouse, D. L. (1989). Layered multiplexing considered harmful. *Proceedings of the 1st International Workshop on High-Speed Networks*, May.
[75] Vinoski, S. (1997). CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, **14**, February.
[76] Wollrath, A., Riggs, R., and Waldo, J. (1996). A distributed object model for the Java system. *USENIX Computing Systems*, **9**, November/December.
[77] Zinky, J. A., Bakken, D. E., and Schantz, R. (1997). Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, **3**, no. (1).

Heterogeneous Data Access in a Mobile Environment – Issues and Solutions

J. B. LIM AND A. R. HURSON

Department of Computer Science and Engineering
220 Pond Lab.
The Pennsylvania State University
University Park, PA 16802
hurson@cse.psu.edu

Abstract

A mobile computing environment involves accessing information through a wireless network connection. The mobile unit may be stationary, in motion, and/or intermittently connected to a fixed (wired) network. As technology advances are made in software and hardware, the feasibility of accessing information "anytime, anywhere" is becoming a reality. Furthermore, the diversity and amount of information available to a given user is increasing at a rapid rate. Current distributed and multidatabase systems are designed to allow timely and reliable access to large amounts of data from different data sources. Issues such as autonomy, heterogeneity, transaction management, concurrency control, transparency, and query resolution have been addressed by researchers for multidatabase systems. These issues are similar to many of the issues involved in accessing information in a mobile environment. However, in a mobile environment, additional complexities are introduced due to network bandwidth, processing power, energy, and display restrictions inherent in mobile devices.

This chapter discusses the fundamental issues involved with mobile data access, the physical environment of mobile systems, and currently implemented mobile solutions. Furthermore, the issues involved in accessing information in a multidatabase environment and mobile computing environment share similarities. Therefore, we propose to superimpose a wireless-mobile computing environment on a multidatabase system in order to realize a system capable of effectively accessing a large amount of data over a wireless medium. We show how one can easily map solutions from one environment to another. This new system is called a mobile data access system (MDAS), which is capable of accessing heterogeneous data sources through both fixed and wireless connections. We will show the feasibility of mapping solutions from one environment to another.

Within the scope of this new environment, a new hierarchical concurrency control algorithm is introduced that allows a potentially large number of users to simultaneously access the available data. Current multidatabase concurrency control schemes do not efficiently manage these accesses because they do not address the limited bandwidth and frequent disconnection associated with

ADVANCES IN COMPUTERS, VOL. 48
ISBN 0-12-012148-4

wireless networks. The proposed concurrency control algorithm—v-lock—uses global locking tables created with semantic information contained within the hierarchy. The locking tables are subsequently used to serialize global transactions, and detect and remove global deadlocks. The performance of the new algorithm is simulated and the results are presented.

1. Introduction

The traditional notion of timely and reliable access to global information in a distributed or multidatabase system must be expanded. Users have become much more demanding in that they desire and sometimes even require access to information "anytime, 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. This information can include data from legacy systems, database systems, data warehouses, information services (i.e. stock quotes, news, airline information, weather information, etc.), and the almost limitless information on the Internet and the World Wide Web. Furthermore, rapidly expanding technology is making available a wide breadth of devices through which access to this enormous amount of diverse

data is possible. For example, the user may access information from a desktop workstation connected to a LAN (Local Area Network), or from a laptop computer via a modem, or from a hand-held device via a wireless connection. All of these devices have different memory, storage, network, power, and display requirements.

Remote access to data is a rapidly expanding and increasingly important aspect of computing. Remote access to data refers to both mobile nodes and fixed nodes accessing a wide variety of data via a network connection characterized by (1) lower bandwidth, (2) frequent disconnection, and (3) higher error rates. Whereas both fixed and mobile node remote connections to the network are made through land-based lines, e.g. a modem and telephone lines, mobile nodes also access data through a wireless medium. This wireless connection results in even more of a severe degradation in the network connection. Furthermore, a mobile node introduces additional complexities such as location dependence, system configuration variations, data placement issues, and long-lived transactions. However, regardless of the hardware device, connection medium, and type of data accessed, all users share the same requirements: timely and reliable access to various types of data. These data types are classified as follows:

(1) Private data—i.e. personal schedules, phone numbers, etc. The reader of this type of data is the sole owner/writer of the data.
(2) Public data—i.e. news, weather, traffic information, flight information, etc. This type of data is maintained by one source, and shared by many.
(3) Shared data—i.e. group data, replicated or fragmented data of a database. A node actually may contribute to maintaining consistency, and participates in distributed decision making with this type of data [3].

Access to various types of data (i.e. private, public, and shared) is not an entirely new concept. Traditional databases have addressed many of the issues involved in accessing these types of data in the form of relational, object-oriented, distributed, federated, and multidatabase management systems (MDBMS). These traditional systems are 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 several disadvantages for traditional database management systems (DBMS). 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.

In order to overcome these shortcomings effectively, a suitable solution must address the following issues:

(1) *Isolation.* A method to deal with the degraded network connection. The solution should also include a means to work offline such that a user can effectively operate if an intentional/unintentional disconnection has occurred. Furthermore, if the connection is too slow or unreliable to work fully online the user may intentionally choose to work offline due to bandwidth restrictions.
(2) *Data integration.* A method to work with a subset of the global data set. This is particularly important for devices with limited resources and capabilities. With a very large amount of data available to the user, the entire set of data cannot be kept locally. Therefore, the user requires a method to choose data from the global set, and use it at local speeds.
(3) *Browsing.* Since there may be an enormous amount of information available to the user, the user should be able to search and look at the available data in an efficient manner. In other words, an efficient method to browse the data.
(4) *Distribution transparency.* The placement of the data and the topology of the network should be designed transparently and to maximize the performance for the overall system. This is particularly important for wireless devices, which have the largest communication cost.
(5) *Limited resources.* The system should be able to accommodate computing devices with limited capabilities. This includes memory, storage, and display deficiencies.
(6) *Data heterogeneity.* With a large amount of data available to the user, name and type differences, and semantic differences/similarities need to be resolved. There are additional difficulties when accessing data in a non-native language. The system should provide a means to address these issues.
(7) *Location transparency.* A higher degree of mobility argues for a higher degree of heterogeneous data access. In particular, a mobile user can potentially access a much wider variety of systems in different locations and can also receive broadcast-type data from various locations due to mobility. Therefore, heterogeneous remote access to data (HRAD) sources is required for a remote access system.

The literature has individually addressed some of the above issues [3, 10, 11, 23, 43, 44]. Current multidatabase systems can effectively handle heterogeneous data access to autonomous systems [10]. Moreover, there are existing mobile applications, which address some of the limited bandwidth issues involved with mobility [15, 17, 22, 23, 26, 27, 28, 44, 46]. However,

a solution does not currently exist that provides an effective means to cope with the issues involved while accessing a massively diverse amount of data over a remote connection. This is particularly true for devices with limited processing capacity and resources.

This chapter is intended to address fundamental issues of mobile data access, the physical environment of mobile systems, and currently implemented mobile solutions. It will compare and contrast the characteristics of multidatabase systems against the characteristics of a wireless-mobile computing environment. In addition, by superimposing a multidatabase system on a wireless-mobile environment it proposes a computational paradigm for effectively accessing heterogeneous data.

In a distributed environment, concurrency is used as a means to increase the throughput and to reduce the response time. Data access in an MDBMS 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 following properties (called ACID properties) hold for each individual transaction [20]:

- *Atomicity.* The operations of a transaction are considered to be atomic; either all operations occur, or none occur.
- *Consistency.* The actions of a transaction taken as a group do not violate any integrity constraints of the database.
- *Isolation.* Although transactions may execute concurrently, each transaction assumes that it is executing alone in the system. In other words, the system hides all intermediate results of a transaction from other concurrently executing transactions.
- *Durability.* Successful completion of a transaction (commit), should guarantee that the effects of the transaction survive failure.

Concurrency control in a mobile data access system (MDAS) involves global transactions under the control of a global transaction manager (GTM) that are correctly serialized to guarantee proper execution. A global transaction is decomposed into sub-transactions and executed as a local transaction at local sites under the control of the local database management system. Proper concurrent execution of transactions should be coordinated such that the interference does not violate the ACID properties. Furthermore, the concurrent execution of transactions should offer a higher throughput than the serial execution of transactions [20]. In an MDAS environment, the concurrent execution of transactions is a more difficult task to control properly. This is mainly due to the inferences among global transactions, inferences

among global and local transactions, local autonomy of each site, and frequent network disconnection.

There has been extensive research performed to maintain the ACID properties in centralized and tightly coupled distributed systems [3, 20]. A primary feature of an MDAS or MDBMS that distinguishes it from a conventional distributed DBMS is the notion of local autonomy. In a conventional, tightly coupled distributed system, each site has limited or no local autonomy. In contrast, local sites in an MDAS operate autonomously, with little or no knowledge of other local DBMSs or the MDAS system.

The degree of autonomy of each local site in an MDAS varies considerably depending upon local and global conditions. Full autonomy refers to the condition where the local site retains full control, and may unilaterally abort a transaction (any local or global sub-transactions). Full autonomy introduces extremely difficult problems for global transaction management in a MDAS. Achieving a high degree of concurrency under full autonomy is not practically possible. However, if local autonomy is somewhat compromised, methods such as locking, time-stamp ordering, ticketing, and serialization graph testing may be used. In general, as the degree of local autonomy decreases, the ability to effectively process transactions in a MDAS becomes easier. Figure 1 illustrates this concept.

Within the scope of this new environment, this chapter will introduce a new hierarchical concurrency control algorithm that is designed to reduce the required communication overhead. The proposed scheme allows for a higher overall throughput and faster response times to users—timely access to data. The concurrency control for global transactions are 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.

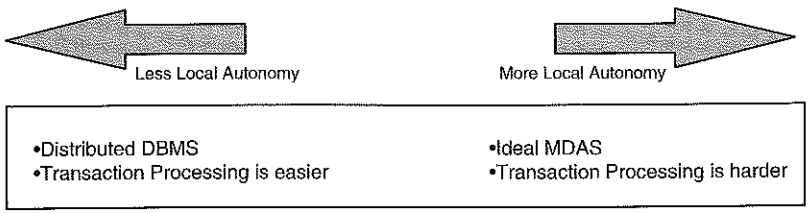

FIG. 1. Degree of mobile support.

(3) The algorithm is designed to work with the summary schemas model (SSM), which is a distributed, hierarchical, multidatabase environment [11]. 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.

Section 2 addresses the necessary background material on mobile systems, while section 3 covers multidatabase systems. Section 4 discusses a new computing environment in which a wireless-mobile computing environment is superimposed on a multidatabase system. The characteristics and issues of this new environment are discussed in this section. Furthermore, the details of a proposed multidatabase system, the summary schemas model (SSM), are discussed. Chapter 5 covers additional background material on concurrency control and a simulation of the proposed concurrency control scheme is introduced. A simulation model is developed to test the effectiveness of the proposed schemes. Subsequently, the results are presented and analyzed. Section 6 concludes the chapter, and discusses future solutions and research directions. Finally, section 7 contains references, a glossary of terms, and an appendix of current wireless/mobile systems.

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. This section will introduce and discuss these topics in detail.

2.1 Physical Environment

2.1.1 Network Architecture

A remote access network consists of a variety of network connections and access devices with varying characteristics as shown in Fig. 2 [3, 16].

- *Fixed, land-based LAN connection.* This type of network connection ranges in speed from 10 Mbps to gigabits per second and usually does not impose any type of restriction on the system. Common fixed LAN connections include ethernet, fast ethernet, gigabit ethernet, and

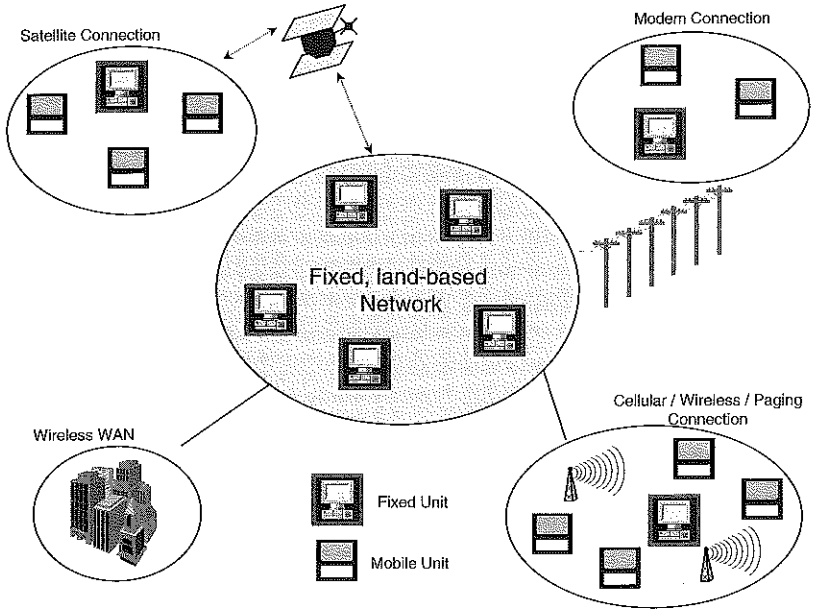

FIG. 2. Network architecture.

asynchronous transfer mode (ATM) [12, 47]. A remote access connection is usually made *to* this type of network.

- *Modem connection.* This is typically what a home/telecommuting/mobile user would use to make a connection. The connection is made directly to a server, or via the Internet through an Internet service provider (ISP). Currently, the majority of these connections are made from a fixed point, such as a home or office. The connection is made with data transfer speeds up to 57.6 kbps [42], and is usually carried over the telephone network. This type of network connection has a much lower bandwidth and is less reliable than a fixed LAN connection. However, with the exception of a fixed connection, it provides the highest performance compared to other network types.
- *Cellular.* A cellular network provides both data and voice services to a user. Traditionally, a cellular network is used with handheld telephones, more recently, it has been used to provide data services to mobile systems. Typical bandwidth ranges from 9.6–19.2 kbps [16]. The cost of using a cellular network is relatively high, and furthermore, existing networking support using cellular technology does not scale well to large numbers of mobile users. Moreover, the use of this technology for data transmission is still in the early stages [16]. Cellular networks, however, do provide very wide coverage, i.e. international coverage is available.

  The geographic area covered by a cellular network is divided into fixed *cells*. Cells are connected to a base station that is attached to a fixed network (e.g. telephone network, data network, etc.). A cell covers a geographical area ranging from one to several square miles. Pico-cells cover a much smaller area (tens to hundreds of meters) and are placed in areas of high traffic and usage. When a user moves from the coverage of one cell to another, a "hand-off" occurs where the connection and control moves to the newly entered cell. This "hand-off" may cause a temporary disconnection, further degrading the reliability of the cellular system.
- *Wireless LAN.* This is a traditional LAN augmented with a wireless interface to provide wireless services within a small geographical area (i.e. up to a few kilometers). The bandwidth of this type of network is limited, ranging from 200 bps to 20 Mbps [30]. There is currently no networking support for any type of wide-area moves.
- *Wide-area wireless network.* These are special mobile radio networks that provide wide coverage and low bandwidth. These types of networks are expensive, provide nationwide data services, and bandwidth is available up to 19.2 kbps. However, there is no support for ubiquitous network connectivity and it is not clear if the systems will scale well for a large number of users [3].
- *Satellite network.* These network systems have been initially designed to carry voice, television, data, messaging, and paging services. They provide very wide coverage (global) at a relatively high cost with a bandwidth ranging from 9.6 kbps to 3 Mbps [41]. Some systems also only operate in a receive-only mode. Currently, the only such operational system is the Hughes Network Systems' DirecPC (the same network used by the DirecTV satellite system for television). Other proposed satellite systems include Motorola's Iridium, Qualcomm's Globalstar, and TRW's Odyssey [33].
- *Paging network.* This network is typically a receive-only system with very low bandwidth. Coverage ranges from a local city to global. AT&T and Motorola currently have two-way paging network systems [32, 41, 49].

In comparison to traditional fixed LANs, remote access connections have lower bandwidths, have higher error rates, and disconnection is much more frequent. Furthermore, due to large variations in bandwidths, there are varying degrees of disconnection that can occur with these types of networks. Table I summarizes these different connection mediums.

TABLE I
PHYSICAL CONNECTION MEDIUMS

Network type	Bandwidth	Transmission	Converge	Availability	Usage cost
Fixed LAN [49, 12]	10 M-Gbps	Symmetrical/Asymmetrical	N/A	High	Low
Modem [42]	up to 57.6 kbps	Symmetrical/Asymmetrical	N/A	Medium	Low
Cellular [49, 16]	9.6–19.2 kbps	Symmetrical	National	Low	High
Wireless LAN [30]	200 bps–20 Mbps	Symmetrical	Several kilometers	Medium	Medium
Wide-area wireless [3]	19.2 kbps	Symmetrical	National	Medium	Medium
Satellite [41, 33]	9.6 k–3 Mbps	Asymmetrical	International	Medium	High
Paging network [49, 41, 32]	less than 9.6 kbps	Asymmetrical	National	Medium	Low

2.1.2 Access Devices

The connection to the network is made through desktop computing devices, portable computers, and portable hand-held devices. The requirements and limitations of these devices vary greatly in terms of computing power, memory capacity, storage space, size, display area, and energy consumption. These devices are summarized in Table II.

Desktop computing devices include workstations, personal computers, and network computing devices. A workstation or desktop personal computer does not have many resource restrictions when connected to a fixed network. These systems can have very large processing capabilities, large memories, large disk drives, and large display areas. The primary restriction pertaining to this type of computing device is due to the remote network connection—typically, a modem when the system is used from a fixed remote location (e.g. home, satellite office, etc.).

A network computing (NC) device is characterized by its low cost and maintenance. As a result, the computer does not have a hard disk, floppy disk, or CD-ROM drive. In order to reduce the cost, the processing power is more limited than a traditional desktop system and memory sizes range from 16MB–128MB. The system must be connected via a network to a server. The NC downloads the operating environment and applications that it runs locally. The prices of these systems range from $500–1500. Currently, Intel/Microsoft, IBM, Oracle, and Sun are manufacturing NCs [37].

Compared to a desktop computer, portable computers have a slightly restricted functionality, but in a smaller package. Memory sizes range anywhere from 8MB–128MB; disk drive capacity can be up to several gigabytes; and typically, they weigh from 4–10 pounds. The display, however, is limited in size (up to 14 inches). In addition, since the system is portable and may not be attached to a non-battery power source, it has a low power requirement. This type of device may use a fixed, modem, or wireless type of network connection.

Portable handheld devices represent the most restrictive and, thus, most challenging device for remote access. First-generation PDAs (personal digital assistant) were handheld messaging units using a wireless connection to send/receive e-mail, small messages and faxes. They had very limited processing capability (equivalent to an Intel 8086), and memory capacity (128K–1MB), small displays, and very strict energy requirements. The Apple Newton and AT&T EO were examples of these types of devices [22]. Due to their limited functionality, the first-generation PDA was not considered very successful. Technological advances have given second-generation PDAs better processing capabilities and larger memories (1MB–32MB). Due to their small size, they still have some of the same restrictions as the

TABLE II
REMOTE ACCESS DEVICES

Device type	Relative computing power	Memory	Storage	Weight	Display size	Energy requirement
Workstation, desktop PC [a,b]	Very high	Gigabytes	Gigabytes–Terabytes (disk based)	< 500 lbs	up to 21 inches	None
Network computer (NC) [37]	High	<128 MB	None	<100 lbs	up to 21 inches	None
Portable computer [c,d]	High	<128 MB	<6 GB (disk or electronic)	4–15 lbs	up to 14 inches	Battery (NiCAD, lithium, NiMH)—typical usage is 1–16 hours per recharge
Portable hand-held computer [e,f]	Low–medium	<32 MB	<500 MB (disk)	4–16 oz	up to 6 inches (640 × 480 pixels)	Battery (alkaline, NiCAD, lithium)— typical usage is up to 8–48 hours per recharge/battery
Pager [g,h]	Very low/none	<16 K	None	1–8 oz	up to 3 inches	Battery (alkaline—typical usage is 30 days)

[a] Sun Microsystems, Technical Specification Sheet for Sun Enterprise 3000 Servers, 1998.
[b] Dell Computer Systems, Technical Specification Sheet for XPS, 1998.
[c] Toshiba Computer, Technical Specification Sheet for Tecra 780DVD, 1998.
[d] IBM Computer, Technical Specification Sheet for Thinkpad 600, 770ED, 1998.
[e] Sharp, Technical Specification Sheet for Mobilon, 1998.
[f] US Robotics, Technical Specification Sheet for Palm III, 1998.
[g] Motorola, Technical Specification Sheet for Advisor, Bravo, Director, PageWriter, 1997.
[h] Motorola, ReFLEX Fact Sheet, 1997.

first-generation PDAs. Second-generation PDAs are more general-purpose computers and are capable of running non-messaging applications (e.g. database, Internet access, etc.). Some examples of these types of devices include the USR PalmPilot, and Windows CE devices.

Finally, pagers represent the most basic form of a handheld remote access device. They have almost no processing power, very limited memory, and very small displays, and are mostly designed to receive telephone numbers, alphanumeric messages or information (e.g. weather, stock, news, etc.). Most pagers are passive devices in that they only receive data and are unable to transmit data [32, 41, 49].

2.2 Mobility

A mobile application must be able to adapt to changing conditions [4]. 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).

2.2.1 *Mobile Support*

A mobile support system can be classified, based upon the level of mobile awareness, into three categories as shown in Fig. 3: mobile-transparent systems, mobile-aware systems, and non-supportive systems. A mobile-transparent system is one where the application is unaware of the environment, and the mobile aspects of the system are hidden from the application. These systems offer backward compatibility with existing applications, and existing applications run without any modifications. However, the major drawback of these systems is the lack of flexibility because they can not adequately adapt to the changing environment of the mobile system. Thus, the result is a degradation of the overall functionality and performance. Furthermore, these systems often require the cooperation of both the user and application [23].

On the other hand, a non-supportive approach offers no system support for mobile applications. The application must handle all of the restrictions imposed by mobility. Although this approach eliminates the need for system support, applications are more difficult to write, incompatibility issues arise, and the software becomes less reusable [3].

The middle ground between these two approaches is the mobile-aware

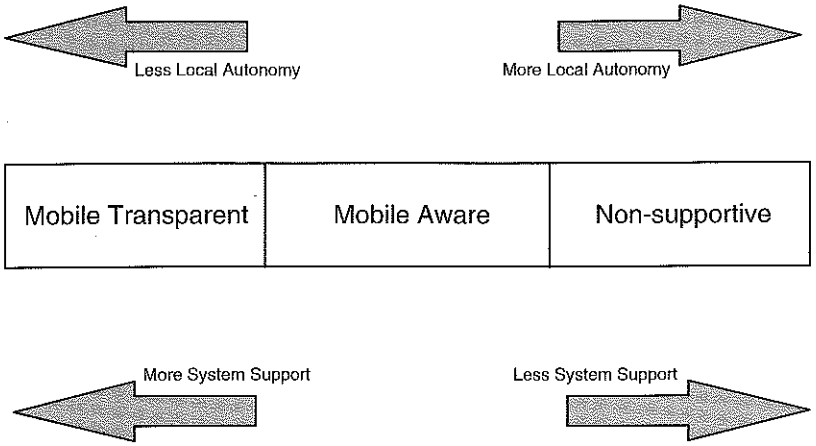

FIG. 3. Degree of mobile support.

environment. Instead of the system masking the environmental conditions from the application, this information is exposed to the applications, enabling them to actually participate in the decision-making process [23]. This approach allows for the system to support an application, as well as for the application to remain autonomous, depending upon environmental conditions, i.e. network, performance, reliability, power, etc.

2.2.2 Autonomy

The level of autonomy required in a mobile system varies with different systems. For mobile-transparent and non-supportive applications, the level of autonomy is fixed. In other words, the level of server dependence and self-reliance is unchanging. However, the level of autonomy required by a mobile-aware system is highly dependent upon the available network bandwidth, the reliability of the connection, available resources in the mobile unit, and the power limitations of the access unit. The application is alert to any changes to these resource restrictions and dynamically adapts accordingly.

The network connection has the largest impact upon the degree of autonomy required by the mobile system. If the system is connected to a land-based network, then the mobile system can rely heavily upon a server. However, as the network connection degrades (in terms of bandwidth and reliability), then the degree of autonomy required by the mobile system increases. The extreme case is when the mobile system is disconnected, and hence full autonomy is required.

2.2.3 Heterogeneous data

The level of access to heterogeneous data increases with mobile systems. A mobile system uses a wireless link as a network connection (albeit with degraded performance). However, because of mobility, there is a high chance for a system to be connected to different host units. This argues for support to access data, which is more heterogeneous than a stationary system. Furthermore, data can be broadcast to mobile users over a wireless medium. As a mobile user moves from one location to another, there is a wider range of broadcast data available to the user. Therefore, a higher

TABLE III
SUMMARY OF MOBILE ENVIRONMENT ISSUES

Mobile environment issue	Description
Site autonomy	Local control over resources and data. The degree of autonomy required depends upon the degree of mobile support offered by the system.
Heterogeneous interoperability	Hardware and software heterogeneity.
Disconnect and weak connection support	A mobile system should provide a means to provide access to data while faced with a disconnection or weak connection.
Support for resource scarce systems	A mobile system should address the inherent limitations of various resource scarce access devices. These include processing, storage, power, and display limitations.
Transaction management and concurrency control	Correct transaction management should satisfy the ACID properties (Atomicity, Consistency, Isolation, and Durability).
Distribution transparency	Distribution of data is transparent to the user.
Location transparency	The location of the data is transparent to the user.
Location dependency	The content of the data is physically dependent upon the location of the user.
System transparency	The user should be able to access the desired data irrespective of the system.
Representation transparency	Representation transparency includes naming differences, format differences, structural differences, and missing or conflicting data.
Intelligent search and browsing or data	The system should provide a means for the user to efficiently search and browse the data.
Intelligent query resolution	The system should be able to efficiently process and optimize a query submitted by the user.

degree of mobility implies that there is a requirement to access a higher degree of heterogeneous data.

Mobile computing is a rapidly expanding technology. A remote access connection introduces many challenges, making the use of mobile devices more difficult. Furthermore, 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 large set of heterogeneous data. Moreover, it must be able to facilitate the bandwidth, resource, and power restrictions of mobile systems. The various issues in accessing data in a mobile computing environment are summarized in Table III.

Multidatabase systems share many of the same characteristics of mobile systems. In the next section, we discuss these characteristics.

3. Multidatabase Systems

Multidatabase systems (MDBMS) are used to maintain, manage, and access a large amount of information. An MDBMS is a global system layer that allows distributed access to multiple pre-existing local database systems. The global layer provides full database functionality and interacts with the local DBMSs at their external user interface. Both the hardware and software intricacies of the different local systems are transparent to the user, and access to different local systems appears to the user as a single, uniform system. The term multidatabase includes federated databases, global schema multidatabases, multidatabase language systems, and homogeneous multidatabase language systems. The typical architecture of an MDBMS is shown in Fig. 4. This section will discuss the various issues involved in order to provide multidatabase functionality to a mobile user, and will show the similarities and differences between the two computational environments. Furthermore, the summary schemas model (SSM) as the underlying heterogeneous multidatabase environment [10] is introduced, and its concepts and structure are discussed.

3.1 Site Autonomy and Heterogeneity

One of the essential features of a multidatabase system is local site autonomy, meaning that each local database management system (DBMS) maintains complete control over local data and resources. Modifications to the global system should not impact the operation of the local system. There are two different classes of operations, global and local. A global operation is performed through the MDBMS on the entire system, and should not affect

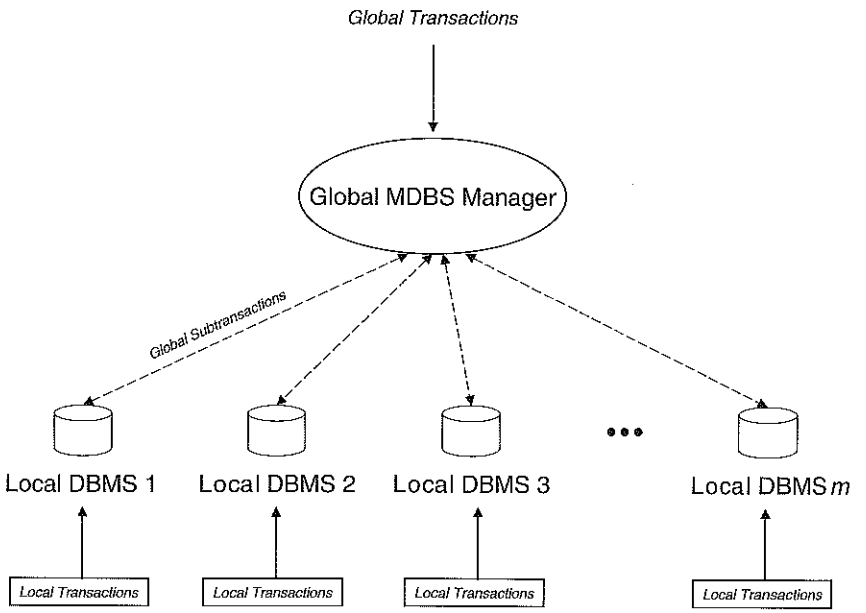

FIG. 4. Architecture of an MDBMS.

the operation of any single, local DBMS. There are three different forms of autonomy [18]:

- *Design autonomy*. When joining an MDBMS, a local DBMS should not require any software or hardware changes. The global MDBMS software should form a "wrapper" around an existing local system, allowing for easy integration into the existing MDBMS. Changes that impact the local system are undesirable, particularly for legacy systems, in which there may be a significant amount of capital invested in existing software, hardware, and user training.
- *Communication autonomy*. During integration into an MDBMS, the local DBMS decides exactly which local data should be available globally. There are two consequences of communication autonomy: (1) the local DBMS may or may not inform the global user about local operations, and (2) a local DBMS is not responsible for synchronization of global operations.
- *Execution autonomy*. A local DBMS executes operations at its site (global or local) in any manner it desires. In other words, a global operation is executed as any "normal" local operation executing at the local site. This implies that the local DBMS has the option of refusing or aborting any operation (global or local).

In addition to autonomy, heterogeneous data access is an important aspect of a multidatabase system. Multidatabase systems are designed to access a multitude of data, irrespective of the system, data type, or particular local access method. Heterogeneity occurs in the form of software or hardware. Support for heterogeneity is a tradeoff between developing and making changes in both hardware and software, and limiting participation. As a result, as the number of local systems and the degree of heterogeneity among these systems rises, the cost of integration into the global MDBMS increases. Furthermore, the global system software must also support any local deficiencies. Particularly with MDBMSs, autonomy and heterogeneity leads to issues in schema integration, query languages, query processing, and transaction management.

3.2 Multidatabase Issues

3.2.1 Schema Integration

When accessing a large amount of heterogeneous data, an MDBMS must address the schema issues related to data representation transparency, system transparency (interoperability), and location transparency. Local databases have their own schema, and the goal of an MDBMS is to provide an integrated schema that presents a logical, global view of the data.

There are many ways to model a given real-world object (or relationships to other objects). Local databases are developed independently with differing local requirements. Therefore, it is a reasonable assumption that a multidatabase system contains many different models or representations for similar objects. The differences in data representation which multidatabases address are [10]:

- *Naming differences.* Different information sources and services may have different conventions for naming data objects.
- *Format differences.* Differences in data type, domain, scale, and precision are types of format differences.
- *Structural differences.* Depending on how an object is used by a database, it may be structured differently throughout local databases.
- *Missing and conflicting data.* Databases that model the same real-world object may have conflicts in the actual data values recorded.

System transparency issues occur in the type of system, network connection, protocol, and data model (i.e. relational, object-oriented, network, hierarchical, etc.) used at the local level of a multidatabase system. The extent of system transparency in a multidatabase system is a tradeoff

between writing translation code (in the form of a gateway or middleware), and limiting local system participation in the multidatabase [10]. Location transparency refers to both the distribution and location of the data. A multidatabase system usually provides location transparency such that a user can submit a query to access distributed objects without having to know the actual location of the object. In a large multidatabase system, a user could not be expected to remember or include the location of objects within a particular query. Distribution transparency hides the distribution of data from the user. In other words, the manner in which the data is distributed across the multidatabase is transparent to the user.

3.2.2 Query Languages and Processing

The basis of global query processing is similar across most MDBMSs. A global query is submitted to the system, and the query is decomposed into a set of subqueries—one for each local DBMS involved in the transaction. Query optimization is used to create an efficient access strategy that includes: (1) which local DBMSs are involved, (2) the operations to be performed at the local DBMSs, (3) integration of intermediate results from the local DBMSs, and (4) the location(s) of the global processing. During the actual execution of the query, the queries may require translation at several levels due to query language and data representational differences at the local DBMS. Furthermore, the distributed nature and execution autonomy of an MDBMS along with different processing capabilities at a local DBMS create additional challenges to an MDBMS. Additional considerations include communication bottlenecks, hot spots at servers, data fragmentation, and incomplete local information [20, 35, 39]. These issues place additional constraints on the query processor. However, despite these difficulties, the query processor must be able to efficiently handle the query processing as well as efficiently manage the global resources.

3.2.3 Transaction Management and Concurrency Control

An MDBMS transaction management scheme must guarantee correctness under all circumstances. By correctness, a transaction should satisfy the ACID properties [20]. Maintaining the ACID properties is desirable in any MDBMS, however, difficulties arise from the requirement to maintain the autonomy of a local DBMS in a multidatabase environment. Local autonomy states that each local DBMS retains complete control over its data. This implies that local operations are outside the control of the global system.

Therefore, the operations in a transaction can be subjected to large delays, frequent or unnecessary aborts, inconsistency, and deadlock.

Finally, concurrency control is an issue which requires consideration in a multidatabase system. Concurrency control in a multidatabase refers to the serializability of global and local transactions. The global system has information regarding global transactions; however, due to local autonomy, the local transactions are not seen at the global level. Therefore, direct or indirect conflicts may arise between local and global transactions. Concurrency control in multidatabase systems should address the issues involved with detecting and resolving these conflicts.

Many of the issues that are involved with accessing data in an MDBMS are summarized in Table IV. Comparing these characteristics to those listed in Table III for a mobile computing environment, one can find many similarities between the two data processing environments. In the next section, we address these characteristics, and discuss the similarities and differences between an MDBMS and mobile computing environment.

TABLE IV
SUMMARY OF MULTIDATABASE ENVIRONMENT ISSUES

Multidatabase environment issues	Description
Site autonomy	Local control over resources and data. Three different forms of autonomy include design, communication and execution.
Heterogeneous interoperability	Hardware and software heterogeneity.
Transaction management and concurrency control	Correct transaction management should satisfy the ACID properties (Atomicity, Consistency, Isolation, and Durability).
Distribution transparency	Distribution of data in the MDBMS is transparent to the user.
Location transparency	The location of the data in the MDBMS is transparent to the user.
System transparency	The user should be able to access the desired data irrespective of the system.
Representation transparency	Representation transparency includes naming differences, format differences, structural differences, and missing or conflicting data.
Intelligent search and browsing of data	The MDBS should provide a means for the user to efficiently search and browse the data contained in the MDBMS.
Intelligent query resolution	An MDBMS should be able to efficiently process and optimize a query submitted to the system.

4. The MDAS Environment

There are similarities in the objectives of effectively accessing data in a multidatabase and a wireless-mobile computing environment. This chapter proposes to superimpose a wireless-mobile computing environment on an MDBMS to realize a system capable of effectively accessing data over a wireless medium. This new system is called a mobile data access system (MDAS). By superimposing an MDBMS onto a mobile computing environment, one should be able to map solutions easily from one environment to another. This section will discuss the issues needed to support this new computing environment. Furthermore, we will discuss the structure of the summary schemas model (SSM), a heterogeneous multidatabase environment [11], and show how the SSM is used as the underlying multidatabase environment in an MDAS.

A summary of the issues facing a multidatabase and those involved in a mobile system is given in Table V. Both systems have autonomy and heterogeneity requirements, where the mobility of a system introduces more complexity. The objective of either system is to provide access to data, where the clients and servers of an MDBMS are typically connected through fixed network connections, and the clients in a mobile environment are typically connected through a wireless connection. Both systems must address the data, software, and hardware heterogeneity issues as discussed in sections 2.2 and 2.3. The larger number of potential data sources, the mobility, and the resource constraints in a mobile environment further complicates

TABLE V
MULTIDATABASE AND MOBILE SYSTEM COMPARISON

	Mobile system	Multidatabase system
Site autonomy	✓	✓
Heterogeneous interoperability	✓	✓
Transaction management and concurrency control	✓	✓
Disconnect and weak connection support	✓	○
Support for resource scarce systems	✓	✗
Distribution transparency	●	○
Location transparency	●	○
Location dependency	●	✗
System transparency	●	○
Representation transparency	●	○
Intelligent search and browsing of data	●	●
Intelligent query resolution	●	●

Key: ✓ Required; ● Desirable; ○ Optional; ✗ Not required

access to the data. The literature has addressed the heterogeneity issues in an MDBMS [10, 11]. However, these issues have not been addressed in a wireless-mobile computing environment. Traditionally, wireless-mobile computing researchers have investigated and addressed wireless connection, mobility, and portability issues, but have been inclined to ignore many of the issues related to heterogeneity. Consequently, the combined solutions from an MBDMS and a wireless-mobile system should be developed to form an integral part of any MDAS.

Autonomy of a system implies that the system should have complete control over the local data and resources, and be able to operate independently. In a multidatabase system, this is referred to as site autonomy, where a local DBMS is autonomous with respect to other systems in the MDBMS. In a mobile system, autonomy refers to the mobile user/application, where the level of autonomy is a function of the available resources (network, processing, storage, etc.). The level of autonomy also varies depending upon the mobile awareness of a particular application, and the support provided by the system. The quality of the wireless/fixed network connection and the processing capacity of the hardware are the primary factors in determining the level of application-autonomy that is required. This type of variable autonomy is only possible if the system and application support this functionality. Some systems may only provide partial application autonomy, or may not even provide any support for this functionality. An MDAS should support both site-level and application-level autonomy.

Schema integration issues include data representation, system, and location transparency issues. As with heterogeneity, these issues have been extensively researched in multidatabase systems [10, 11]. A primary goal of an MDBMS is to present the user with an integrated, global schema of the data. However, in a wireless-mobile computing environment, researchers have overlooked the importance of schema integration in a data access system. Particularly since mobility tends to increase the degree of heterogeneous data available, an MDAS must address schema integration issues in order to present the user with a viable solution for accessing heterogeneous data. Furthermore, mobility introduces an additional challenge in that it may be desirable to have location *dependence* when accessing data. In such instances, the content and representation of the data could actually depend upon the location of the accessing the data.

Query processing issues are well understood in an MDBMS. A global query is submitted to the system, and the query is decomposed into a set of sub-queries, one for each local DBMS involved in the transaction. In a mobile environment, where the processing power, storage, and energy may be restricted, query processing is non-trivial. If a mobile unit has sufficient resources to perform the query processing, then the query in the MDAS could be processed and executed similar to a query in an MDBMS. However, if the resources are limited, then the processing should be performed by a fixed, more resourceful computing device in the MDAS. One of the disadvantages of this method is that there may be an increase in the network traffic, which poses a problem in a wireless connection. Different strategies to address these issues include object-oriented designs, dynamic adjustment to bandwidth changes, data distillation, and query processing on fixed network devices [10, 23, 44].

Effectively accessing the data in a heterogeneous environment may require an efficient means of searching/browsing the data, in addition to an efficient mechanism to resolve and process a user's query. In a mobile environment, this may be more difficult to realize due to network, storage, processing power, and energy restrictions. Similar to query processing, the processing could be performed by a fixed, more resourceful computing device in the MDAS if the local host does not have the resources to search/browse data. Furthermore, network traffic increases depending upon the storage capacity of the mobile unit. The lower the storage space the local unit contains, the more the likelihood increases of generating network traffic. In other words, if the local node could store more information and data about the global schema and data, additional local processing (and hence less network traffic) could be achieved.

Transaction processing and concurrency control is an important, yet extremely challenging aspect of data processing. MDBMS researchers have been faced with the problem of maintaining serializability for global transactions. The problem of maintaining serializability in a multidatabase is complicated by the presence of local transactions that are invisible at the global level. There are two methods used to maintain serializability in an MDBMS:

1. *Bottom-up approach.* The global serializability is verified by collecting local information from the local DBMSs and validating the serialization orders at the global level. The global scheduler is responsible for detecting and resolving incompatibilities between global transactions and local serialization orders. Optimistic concurrency control mechanisms are usually used with the bottom-up approach. This optimistic nature allows for a higher degree of concurrency among transactions. However, the higher throughput is achieved at the expense of lower resource utilization and more overhead due to rollbacks from failed transactions.
2. *Top-down approach.* The global scheduler is allowed to determine the serialization order of global transactions before they are submitted to the local sites. The local DBMS must then enforce this order at the

local site. This method is a pessimistic approach, and subsequently leads to a potentially lower degree of concurrency. It forces the global order on the local schedulers. Consequently, runtime decisions regarding the ordering of transactions is not needed.

In an MDAS environment, the system should be able to provide global serializability and local autonomy to a user using a wireless connection. The restrictions imposed by a wireless connection have led to the use of optimistic concurrency control schemes in mobile-wireless environments [22, 25, 36, 39]. In addition, an application in an MDAS may be required to use both weak and strong consistency rules, where the application is required to adapt to changing environmental conditions. An operation uses weak consistency guidelines when data in a write operation is updated or written without immediate confirmation, and data in a read operation is based upon an approximately accurate value. In MDBMSs, weak consistency is used to increase global transaction throughput. In an MDAS, weak consistency may be required due to disconnection and/or a weak network connection.

Mobility and some of its consequences—e.g. disconnection and weak connections (communication restrictions), processing, storage, display, and energy restrictions—introduce additional complexities when a user accesses data. A local cache and prefetching in a mobile unit has been extensively used to address the problems associated with disconnection and weak connections. The idea is that when a disconnection occurs, the mobile unit operates in an autonomous state while performing operations on the local cache. When the connection is re-established, a resynchronization between the cache in the local unit and the server occurs [4, 15, 47]. The use of various prefetch schemes has been used to ensure that the required data is available in the cache during a disconnection [4, 36, 44, 46, 51]. Additionally, some type of queuing mechanism is usually provided in order to perform operations on data that may not be contained in the cache [4]. Predictive schemes, where the system is actually able to anticipate a disconnection, are used to lessen the impact of a disconnection. Finally, broadcasting of data on wireless channels has been suggested in order to reduce network traffic [52].

Finally, processing power and display limitations in a mobile unit introduce additional challenges to an MDAS. Offloading the processing performed on the local unit to fixed hosts is commonly used in wireless-mobile environments. Data distillation is commonly used to address the display network limitations of a mobile unit. Many mobile units are not capable of displaying multimedia data (video, images, sound, etc.). Data distillation is a process where incoming data is "distilled," or processed, such that only

TABLE VI
SUMMARY OF ISSUES AND SOLUTIONS FOR AN MDAS

Characteristics	Issues	Solutions
Site autonomy	Autonomy is required in an MDAS system.	Provide both site-level and application-level autonomy.
Heterogeneous interoperability	Heterogeneous interoperability is required in an MDAS system.	Use traditional methods for heterogeneity from MDBMSs.
Transaction management and concurrency control	Provide global serializability to transactions.	Use a bottom-up approach with optimistic concurrency control.
Disconnect and weak connection support	A wireless medium results in lower bandwidth and disconnections.	Local cache, prefetching, and broadcasts.
Support for resource scarce systems	Limited processing power, storage, energy, and display.	Object-oriented design, multi-tiered architecture, offload processing to fixed hosts, data distillation, and broadcasting.
Distribution transparency Location transparency Location dependency System transparency Representation transparency	Schema integration issues, and greater impact due to mobility.	Use traditional methods for schema integration from MDBMSs.
Intelligent search and browsing of data	Limited processing power, storage, energy, and display.	Reduction of local data storage requirements, object-oriented design, multi-tiered architecture, and offload processing to fixed hosts.
Intelligent query resolution	Limited processing power, storage, energy, and display.	Object-oriented design, multi-tiered architecture, offload processing to fixed hosts, and data distillation.

portions of the data that the unit is capable of displaying are shown on the screen. Furthermore, if network bandwidth is limited, data distillation is used to reduce the network traffic by distilling video, images, or sound. In order to address the limitations inherent in a mobile unit, an MDAS should use some or all of these aforementioned methods. A summary of the issues and possible solutions in an MDAS are given in Table VI.

4.1 Summary Schemas Model for Multidatabase Systems

Accessing a heterogeneous multidatabase system is a challenging problem. Multidatabase language and global schema systems suffer from inefficiencies and scalability problems. The SSM has been proposed as an efficient means to access data in a heterogeneous multidatabase environment [11]. The SSM primarily acts as a backbone to a multidatabase for query resolution. It uses a hierarchical meta structure that provides an incrementally concise view of the data in the form of summary schemas. The hierarchical data structure of the SSM consists of leaf nodes and summary schema nodes. Each leaf node represents a portion of a local database that is globally shared. The summary schema nodes provide a more concise view

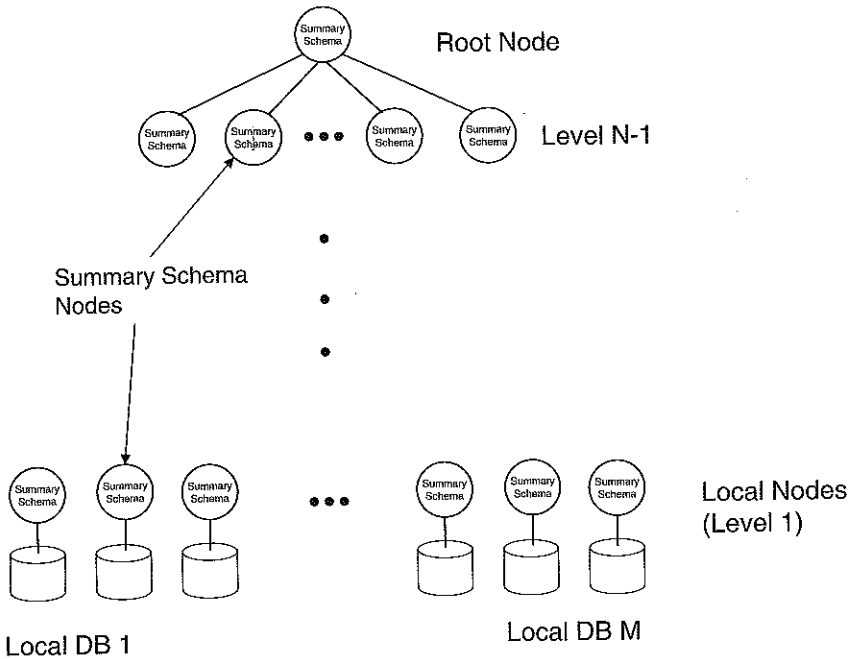

FIG. 5. Summary Schemas Model, N levels, and M local nodes.

of the data by summarizing the schema of each child node. Figure 5 depicts the architecture of the SSM. The terms in the schemas are related through synonym, hypernym and hyponym links [11]. Synonyms are words with a similar definition, and are used to link terms within the same level of the hierarchy. A hypernym is a word that has a more comprehensive and general definition, whereas a hyponym is a word that has a more precise definition. Hypernyms and hyponyms are used to establish links between terms of parent and child nodes.

The SSM intelligently resolves user queries [11]. When the user knows the precise access terms of a local database, the query is directly submitted to the corresponding local data source. However, when the precise access terms and/or location of the data not known to the user, the SSM processes the query in the user's own terms and submits a global query to the system. A simulation and benchmark of the SSM model was performed, and the benefits are shown in [10, 11, 14]. 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 using the SSM for the imprecise queries (access terms and locations are unknown). The results showed that the intelligent query processing of the SSM and an exact query incurred very comparable costs (i.e. there were only small overhead costs to using the SSM) [11]. Interestingly, using intelligent SSM query processing actually outperformed an exact query in certain circumstances [14]. These results are very relevant to an MDAS in that a user would most likely require this functionality while accessing heterogeneous data sources.

The SSM provides several benefits to the user over traditional multidatabase systems, which 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, compared to the requirements of a global schema, are drastically reduced by up to 94% [14]. Subsequently, the SSM 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 meta data structure could be stored locally, which would still provide a global view (albeit less detailed) of the system.
- The SSM can be used to browse/view global data in the multidatabase system. 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. Moreover, for resource limited devices in an MDAS, portions of the upper levels of the SSM could be used to provide a global view (albeit less detailed) of the system, which could be used for browsing/searching.

5. Transaction Management and Concurrency Control

A transaction is one of the fundamental concepts in a DBMS. A transaction is essentially a set of read and write operations performed on data items. It is atomic in that every operation in the transaction either completes, or does not complete. In other words, transactions comprise read and write operations terminated by either a commit operation, or an abort operation.

In order to increase the overall throughput, transactions are executed concurrently. The operations of different transactions can be interleaved as long as each transaction adheres to the ACID properties. The incorrect interaction of operations between two concurrent transactions is the only manner in which an inconsistency can occur in the system. Two read operations by different transactions, however, do not cause inconsistencies because the data values are not changed. The inconsistencies that occur are due to three anomalies [20]:

(1) *Lost update*. An update by one of two transactions is lost. For example, in a lost update, a write operation by transaction T1 is ignored by transaction T2. Transaction T2 writes based upon the original value of the data before transaction T1 altered the data.
(2) *Dirty read*. The final version of a read operation is inconsistent because it is not the final version of another transaction. For example, transaction T2 writes to a data item, then transaction T1 reads the data, and then transaction T2 makes further changes to the data.
(3) *Unrepeatable read*. Two read operations return different values. In this case, transaction T1 reads a data object, transaction T2 changes the data, followed by transaction T1 reading the altered value of the data.

These three anomalies are the basis of concurrency control; if they can be prevented, then the transactions may be interleaved in any manner [20]. This section introduces some additional background material for concurrency control. Subsequently, a new concurrency control algorithm is introduced and discussed.

5.1 Multidatabase and MDAS Transaction Processing Model

There are two basic transactions in an MDBMS:

- *Local transactions*. Transactions that are executed at the local site on local data. These transactions are managed only by the local system, outside the control of the MDBMS.
- *Global transactions*. Transactions that are submitted through a global interface executed under the control of the MDBMS. A global transaction consists of a number of potential sub-transactions executing at multiple local sites. Each sub-transaction, though, appears as a regular local transaction to the local DBMS.

The autonomy requirement of local databases in an MDAS introduces additional complexities in maintaining serializable histories because the local transactions are not seen at the global level. There are two types of conflicts that may arise due to the concurrent execution of transactions—*direct* and *indirect* conflicts. A direct conflict between two transactions T_a and T_b exists if, and only if, an operation of T_a on data item x (denoted $O(T_a(x))$) is followed by $O(T_b(x))$, where T_a does not commit or abort before $O(T_b(x))$, and either $O(T_a(x))$ or $O(T_b(x))$ is a write operation. An indirect conflict between the two transactions T_a and T_b exists if, and only if, there exists a sequence of transactions $T_1, T_2, \ldots T_n$ such that T_a is in direct conflict with T_1, T_1 is in direct conflict with $T_2, \ldots,$ and T_n is in direct with T_b [8].

5.1.1 Indirect Conflict Example

Indirect conflicts are due to the execution of local transactions that can not be seen at the global level. The following example illustrates this problem. Suppose we have two sites and four transactions. Site 1 contains data items s and t; Site 2 contains data items x and y. There are two global transactions T_{G1} and T_{G2}, and two local transactions T_{L1} and T_{L2}. A w indicates a write operation, and an r indicates a read operation on a data item. The following operations represent the four transactions:

$$T_{G1}: w(G1, s), w_{G1}(G1, y)$$
$$T_{G2}: r(G2, x), r(G2, t)$$
$$T_{L1}: w(L1, t)\ r(L1, s)$$
$$T_{L2}: r(L2, y)\ w(L2, x)$$

Assume that a history is generated where T_{G1} has executed $w_{G1}(s)$ and T_{G2}

has executed $r_{G2}(x)$. Next, the two local transactions are executed. Finally, T_{G1} executes $w_{G1}(y)$ and T_{G2} executes $r_{G2}(t)$. This results in the following histories at site 1 and site 2:

Site 1: $w(G1,s)$, $w(L1,t)$, $r(L1,s)$, $r(G2,t)$
Site 2: $r(G2,x)$, $r(L2,y)$, $w(L2,x)$, $w(G1,y)$

The resulting local serialization graphs of each local site are both acyclic as shown in Figs 6 and 7. However, the resulting global serialization graph shown in Fig. 8 contains a cycle. Therefore, although there are no direct conflicts between global transactions, and each serialization graph at the local site is acyclic, a cycle is introduced in the global serialization graph due to the local operations. Since the GTM usually does not have any information regarding the transactions running at the local system, these types of indirect conflicts are very difficult to detect [8]. The reason that indirect conflicts do not occur in distributed database systems is that these solutions assume local sites are tightly coupled, homogeneous systems. Each site is also assumed to use the same concurrency control scheme along with the sharing of local control information, i.e. WFGs, serialization graphs, etc. It should also be noted that in the above example, if every site were to use a locking scheme for serialization, then a global deadlock would result.

5.1.2 *Direct Conflict Example*

The previous example illustrated the problem which arises with indirect conflicts from local transactions. It is possible for direct conflicts between global sub-transactions also to result in inconsistent data. The following

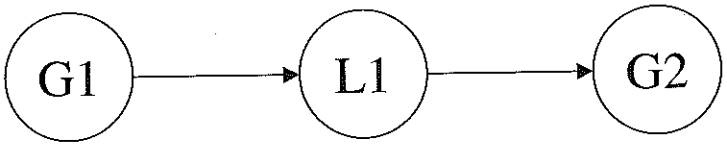

FIG. 6. Local conflict graph of site 1.

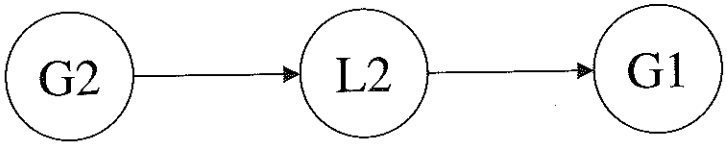

FIG. 7. Local conflict graph of local site 2.

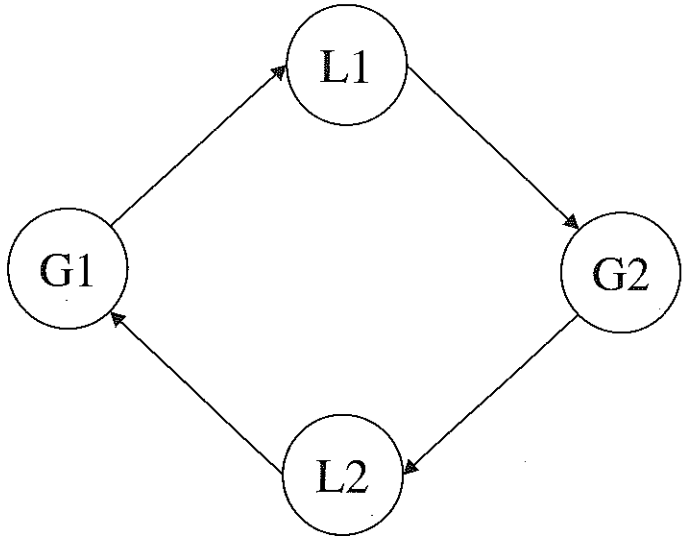

FIG. 8. Global conflict graph.

example of two global transactions illustrates this problem. Suppose we have two global transactions G1 and G2, operating at two sites. Site 1 contains the data item x, while site 2 contains data item y. The following operations represent the two transactions:

$w(G1,x)$ and $w(G2,x)$
$w(G1,y)$ and $w(G2,y)$

The following histories are generated at site 1 and site 2:

Site 1: $w(G1,x)$ and $w(G2,x)$
Site 2: $w(G2,y)$ and $w(G1,y)$

The local serialization graphs are acyclic for each local site as shown in Figs 9 and 10. Figure 11 shows that the global serialization graph contains a cycle. This cycle is due to a direct conflict between the two global transactions, G1 and G2.

FIG. 9. Local conflict graph of site 1.

FIG. 10. Local conflict graph of site 2.

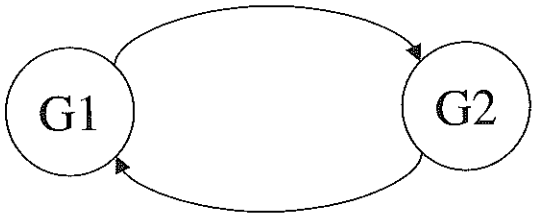

FIG. 11. Global conflict graph of a direct conflict.

5.2 MDBMS Serializability Solutions

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 MDBMS 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 MDBMS needs to address both direct and indirect conflicts between global transactions. The remainder of this section describes several concurrency control algorithms that have been developed for an MDBMS.

5.2.1 *Forced Conflicts under Full Autonomy*

When a local site operates under full autonomy, there is no knowledge of the underlying concurrency control scheme used at the local DBMS. Furthermore, modifications to the local MDBMS to facilitate global concurrency control are not permitted under any conditions. This includes the use of the two-phase commit (2PC) or any other distributed commit protocol that requires a "prepared state."

The only known practical solution to multidatabase concurrency control that operates effectively under these stringent requirements is the forced conflict method [19]. Global serializability is achieved by forcing conflicts among global transactions through special data items called tickets. Global transactions at each site are required to read, increment, and subsequently write the new value of the ticket. The main disadvantage to this method is that the ticket operation may lead to numerous false aborts when using an aggressive policy, or lower concurrency when using a conservative policy [8].

5.2.2 *Site Graph Algorithm*

The site graph algorithm takes a conservative approach towards addressing MDBMS concurrency control [6]. This algorithm is somewhat similar to the lock-per-site algorithm [2]. The site graph algorithm maintains an undirected graph to execute concurrent transactions. Nodes in the graph represent transactions and sites that contain the data items accessed by a global transaction. The edges in the graph represent the sites spanned by a global transaction. For each global transaction, edges are added to the graph between the transaction and each site node in which the transaction participates. If a cycle in the site graph is detected, the transaction that caused the cycle is delayed until the graph becomes acyclic.

A transaction node and its associated edges can only be safely removed from the graph if the transaction aborts. Because conflicts may still occur after a transaction commits, a committed transaction node can only be safely removed from the graph if there are no paths to an uncommitted transaction [5]. This algorithm suffers from lower concurrency because a cycle in the graph does not necessarily imply an existing conflict, but it is used as a method to prevent possible conflicts.

5.2.3 *Locking Schemes*

When the local sites use locking schemes for concurrency control (i.e. most commercial systems), deadlocks resulting from either direct or indirect conflicts are the major challenge at the multidatabase level. A deadlocked global transaction could potentially block global as well as local transactions in *multiple sites* and, hence, will block resources and potentially reduce overall throughput at both global and local transaction levels.

It has been proven that if all local sites produce strict schedules and the transaction uses an atomic commit protocol then the resulting global schedule history is guaranteed to be serializable [7, 8]. It has been similarly shown that if all local sites produce at least recoverable schedules then it is sufficient to synchronize only the commit operations of the global transaction in order to ensure global serializability [6].

There are several multidatabase concurrency control schemes such as potential site graph locking [6], altruistic locking [8], site-graph method [6], ITM [54], and 2PC Agent [52] that use locking as the basis for concurrency

control. When used in an environment where local sites produce strict histories, these algorithms result in serializable schedules by either preventing or detecting deadlocks arising between conflicts.

The information about the status of a local lock (pending or granted) of a data item is obtained from the acknowledgement of a transactional operation at the local site. Main disadvantages are that (1) there is no way to distinguish between direct conflicts and indirect conflicts, and (2) every operation that is sent to a local site, from a global coordinator, must be individually acknowledged.

As discussed before, a multidatabase transaction manager must process global transactions between loosely coupled, heterogeneous systems while preserving as much local autonomy as possible. Minimizing communication overhead in the system is also an important factor, particularly in a distributed environment consisting of mobile units and wireless communication. Furthermore, the cost of a multidatabase transaction processing is based upon parameters that are significantly different from the traditional distributed database systems and, hence, the tradeoffs between aggressive and conservative concurrency control schemes could be a determining factor. The proposed MDAS concurrency control algorithm, discussed in the next section, is intended to address these objectives.

5.3 Concurrency Control for an MDAS

The proposed v-locking algorithm uses a global locking scheme (GLS) in order 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. This is impractical due to the delay and amount of communication overhead involved.

In the proposed algorithm, the MDAS is a collection of summary schema nodes and local databases distributed among local sites. A communication network as previously shown in Fig. 5 interconnects the SSM nodes and databases. The only difference in the MDAS model is that any communication link can be modeled as a wireless link. A transaction consists of a collection of read (r), write (w), commit (c), and abort (a) operations. The MDAS software is distributed in a hierarchical structure similar to the hierarchical structure of the SSM. Subsequently, transaction management is performed at the global level in a hierarchical, distributed manner.

The motivation behind using a hierarchical transaction management organization is due to the hierarchical structure of the SSM and the fact that such

an organization offers higher performance and reliability [11, 31]. 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 and mapped into sub-transactions by the SSM structure [11]. The resolution of the transaction also includes the determination of a coordinating node within the structure of the SSM—the coordinating node being the lowest summary schema node that semantically contains the information space manipulated by the global transaction. The 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.

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

5.3.1 Algorithm

The semantic information contained in the summary schemas is used to maintain global locking tables. Since each summary schema node contains the semantic contents of its children schemas, the "data" item being locked is reflected exactly or as a hypernym term in the summary schema of the

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. 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. Higher reliability at the expense of lower throughput is the direct consequence of the application of semantic contents rather than an exact contents for an aggressive approach.

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 (acknowledgement signals) between the GTM and local sites, based on the network traffic and/or a threshold value. The number of acknowledgements that are performed varies from one per operation to only a single acknowledgement of the final commit/abort of the transaction. 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. This effect is illustrated in Fig. 12. The extent of this incorrect detection of deadlocks is studied in the simulation.

The pseudo-code for the global locking algorithm is given in Figs 13 and 14. Figure 13 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 acknowledgement signal received, the corresponding edge in the graph is marked as exact. In the second case, where each write operation generates an acknowledgement signal, for each signal only the edges preceding the last known acknowledgement are 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 acknowledgement signals enables us to detect cycles in the wait-for-graph.

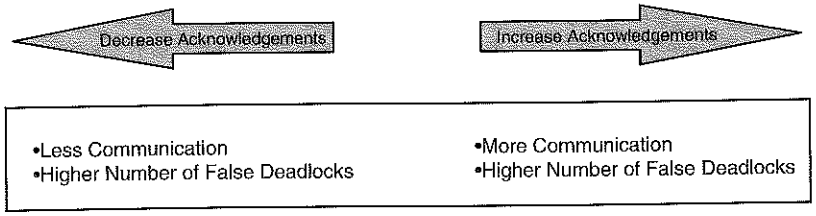

FIG. 12. Frequency of acknowledgements.


```
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 as exact or imprecise
                Else
                        Queue the data item into the locking table
                        Enter edge into wait-for-graph and mark 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
                Repeat until write
                        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 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 acknowledgement of write
                        Mark current and all proceeding read operations as acknowledged
                End repeat for write
        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 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. 13. 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. 14. Global locking algorithm—detection of cycles.

Figure 14 shows how to detect cycles in the wait-for-graph based on the depth first search (DFS) policy [13]. 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 acknowledgements 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 acknowledgement 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.

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 [9]. 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.

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. The pseudo-code for the extended algorithm is given in Fig. 15.

5.3.1.1 Handling Unknown Local Data Sources

Finally, the issue of handling the local "black box" site in which nothing is known about the local concurrency control scheme is addressed. This algorithm is extended to handle this case. Since nearly every commercial database system uses some form of 2PL, this case will only comprise a small percentage of local systems. Therefore, for this special case, the algorithm merely executes global transactions at that site in a serial order. This is done by requiring any transaction involving the "black box" to first 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).

This section has examined some additional concurrency control issues in an MDAS environment. A new hierarchical concurrency control scheme was introduced and discussed. The scheme uses global locking scheme in order to serialize conflicting operations of global transactions. The semantic information in the summary schemas is used to maintain the global locking tables, and reduce the communication required by the GTM in order to serialize the global transactions.

```
Repeat every chktime threshold
        Check for cycles in implied wait-for-graph /* depth first search (DFS) used for cycle
                detection */
        For each cycle detected
                If time_elapsed > pcg_time for all transactions in cycle
                        /* victim is chosen based upon "progress" */
                        Choose victim and break deadlock
                Else
                        Continue to wait
End repeat
```

FIG. 15. Global deadlock detection—potential graph algorithm.

6. Evaluation of Proposed Algorithm

The simulation models an MDAS environment and tests various aspects of the algorithms—v-locking and p-caching. Through the simulation, the proposed schemes are validated and the results are analyzed and discussed.

6.1 V-Locking Concurrency Control Scheme

6.1.1 Simulation

The performance of the proposed algorithm is evaluated through a simulator written in C++ using CSIM. The simulator for the v-locking algorithm 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 our algorithm against the site-graph, potential conflict graph, and the forced conflict algorithms.

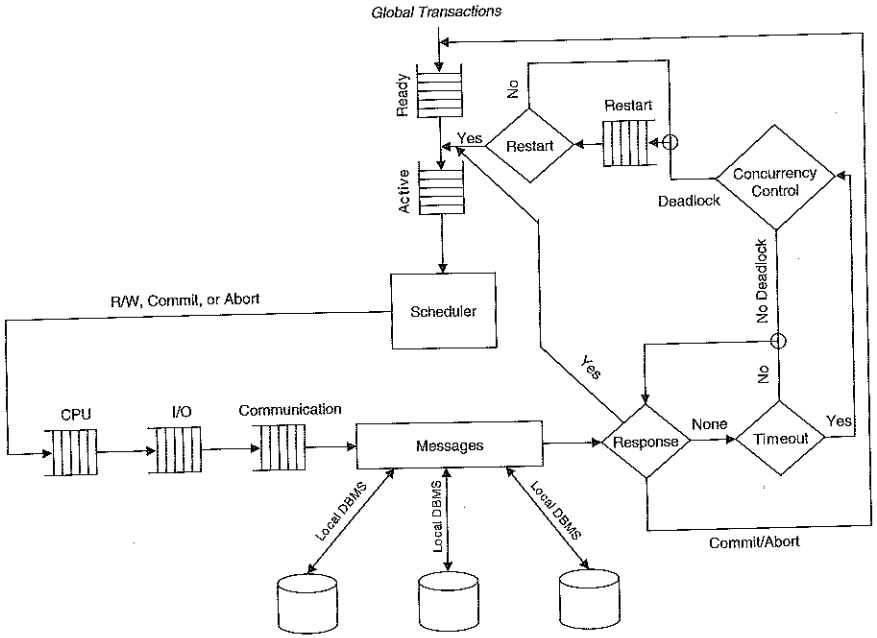

FIG. 16. Transaction flow at the global level.

The MDAS consists of both local and global components. The local component comprises local DBMS systems, each performing local transactions outside the control of the MDAS. The global component consists of the hierarchical global structure, performing global transactions executing under the control of the MDAS. Figure 16 depicts the global structure, whereas Fig. 17 shows the flow of operations in the local components.

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 is first generated, and subsequently enters the active queue. 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 communicates the operation(s) to the local system based upon the available bandwidth. When acknowledgements or commit/abort signals are received from the local site, the algorithm determines if the transaction should proceed, commit, or abort. If a global commit is possible, then a new global transaction is generated and placed in the ready queue. However, if a deadlock is

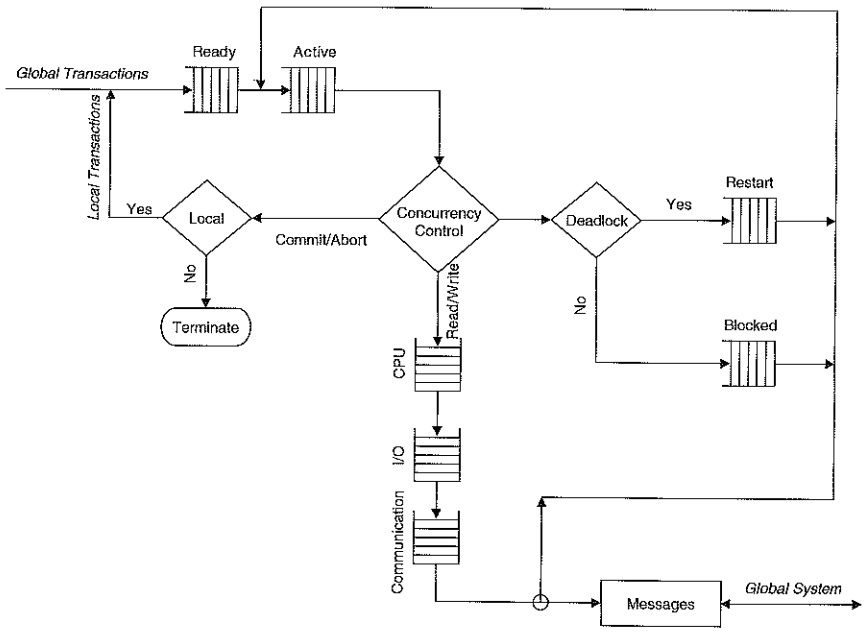

FIG. 17. Local transaction.

detected, or an abort message is received from a local site, then the transaction is aborted at all sites and the global transaction is placed in the restart queue. After a specified time has elapsed, the aborted transaction is again placed on the active queue.

At the local sites, there are a fixed number of active local transactions. The active transactions consist of both local transactions and global sub-transactions. 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 sub-transactions, is communicated back to the GLS. The acknowledgement for these transactions is communicated back based upon the available communication bandwidth. If a lock is not granted, the system checks for deadlocks and will either place the transaction 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 sub-transactions, an abort or commit signal is communicated back to the GLS and sub-transaction terminates.

6.1.2 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 5000 time units, and the 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 sub-transaction (read, write, commit, or abort) may require data and/or acknowledgements 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 VII, along with their default values.

The local systems perform two different types of transactions, local and global. Global sub-transactions are submitted to the local DBMS and appear as a local transaction. Local transactions are generated at the local sites and consist of a random number of read/write operations. The only difference between the two transactions is that a global sub-transaction will communicate with the global system, whereas the local transaction terminates upon a commit or abort. The number of local transactions, which can be varied,

TABLE VII
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	0.005 sec
The service time for the I/O queue	0.010 sec
The service time for each communicated message to the local site	0.100 sec
The number of messages per operation (read/write)	2

affects the performance of the global system. In addition, the local system may abort a transaction, global or local, at any time. If a global sub-transaction 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 VIII along with their default values. Both the global and local systems are modeled similar to models used in [1, 9].

6.1.3 Performance Results

The performance of the algorithm (V-Lock) is evaluated relative to the number of completed global transactions, the average response time, as well as the communication utilization at each local site. In addition, the simulator compares and contrasts the proposed algorithm against the potential conflict graph method [7], site-graph method [2], and the forced conflict

TABLE VIII
LOCAL PARAMETERS

Local 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	0.005 sec
The service time for the local I/O queue	0.010 sec
The service time for each communicated message to the MDAS	0.100 sec
The number of messages per operation (read/write)	2

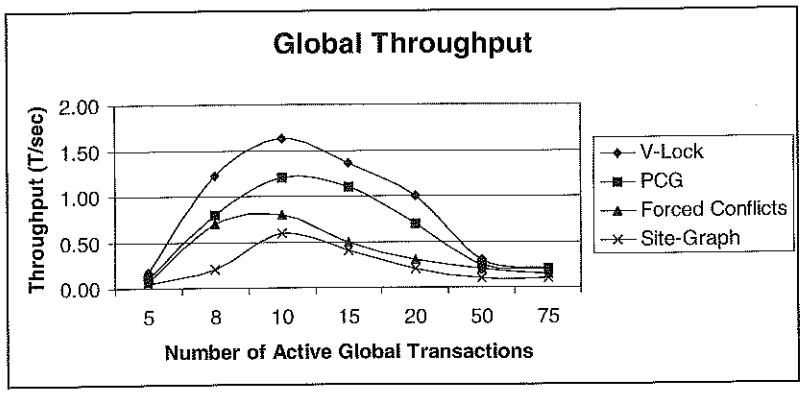

FIG. 18. Comparison of different concurrency control algorithms.

method [19]. Figure 18 shows the results. 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 occurs at a multi-programming level approximately equal to ten. 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.

Figure 19 shows the relationship between the global throughput and the number of sites in the MDAS. The number of sites was varied from 10 sites to 40 sites. The throughput decreases as the number of sites is increased. By fragmenting the data across more sites, the probability of a global transaction spanning over more sites also increases. This increases the likelihood of a conflict occurring and consequently, the lower throughput.

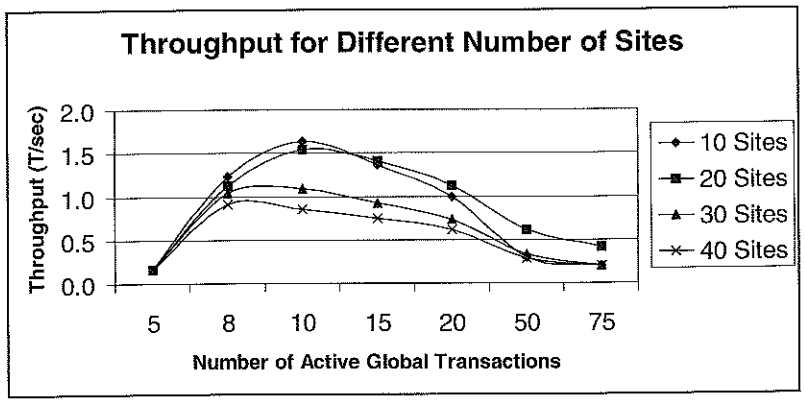

FIG. 19. Global throughput varying the number of local sites.

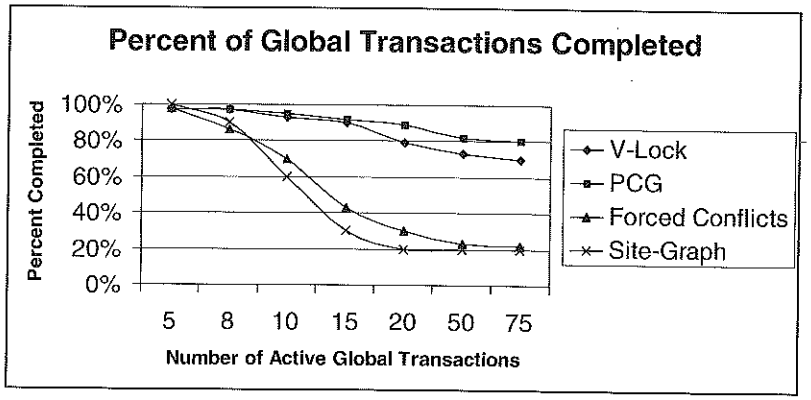

FIG. 20. Comparison of the percent of completed global transactions.

The simulator also measured the percentage of completed transactions during a certain period of time for the V-Lock, PCG, forced conflict, and site-graph schemes. Figure 20 shows the results. 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. On separate simulation runs, the simulator measured, compared, and contrasted the response time for various schemes. Figure 21 shows the results. The two locking algorithms have a

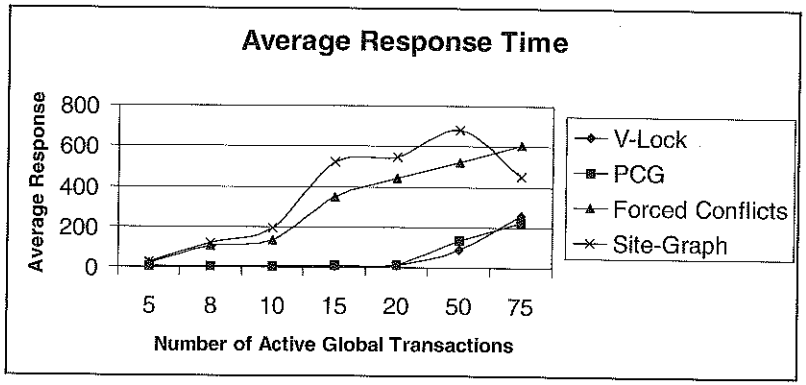

FIG. 21. Average response time.

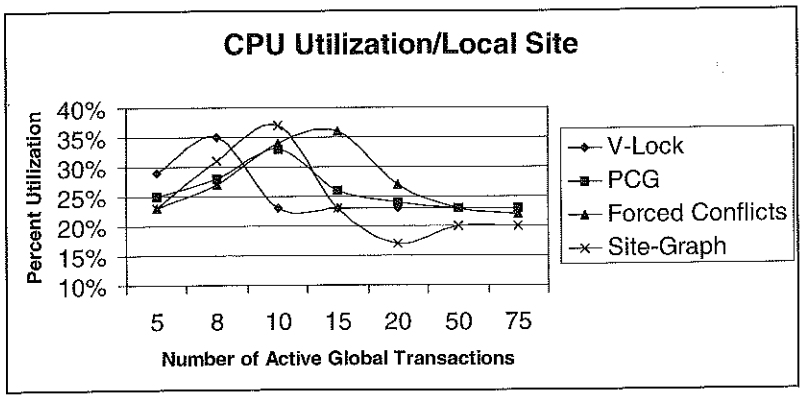

FIG. 22. CPU utilization for each local site.

much better response time than the forced-conflict and site-graph algorithms. The v-locking algorithm has the best response time, and performs better than PCG, particularly for a large number of users. As expected, as the number of concurrent users increases, the response time increases.

Finally, the resource utilization (communication, I/O, and CPU) is compared and contrasted. Figures 22, 23, and 24 show the results. From Fig. 22 it can be concluded that each scheme utilizes approximately 20–25% of the CPU time. Disk utilization generally follows the throughput of the system, with a range of 40–60% utilization under high concurrency (Fig. 23). Finally, the communication utilization is near 100% at peak throughput, and decreases slightly as the number of concurrent transactions increases

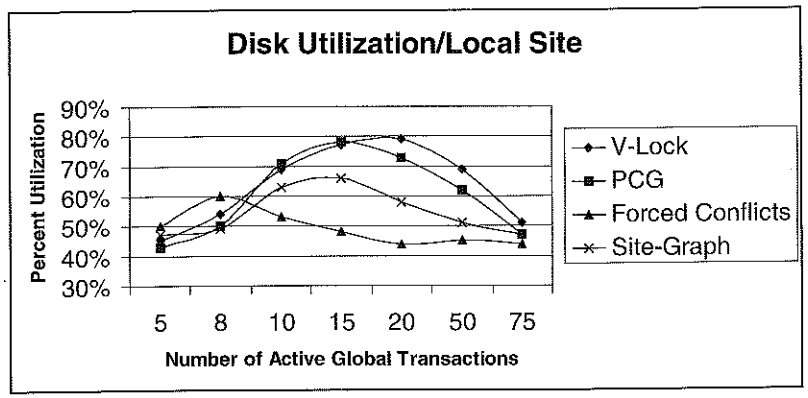

FIG. 23. Disk utilization for each local site.

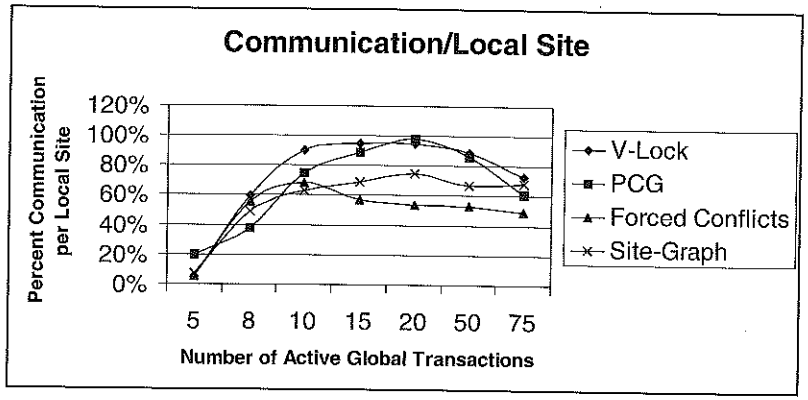

FIG. 24. Communication utilization for each local site.

(Fig. 24). It is easy to determine from this graph that the communication requirements for the v-locking algorithm represent the bottleneck of the system.

7. Conclusions and Future Directions

7.1 Conclusion

The requirements of an "anytime, 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. With advances in technology, it is now possible to access globally shared data through wireless connections via a diverse number of access devices. These devices differ in computing, energy, display, and network requirements.

By superimposing a wireless-mobile computing environment on a multidatabase system, many of the objectives involved in accessing heterogeneous data sources through a wireless medium are addressed. This new class of computing environment is called an MDAS. Furthermore, the similarities and differences in the characteristics of both a mobile environment and a multidatabase system were individually discussed. The characteristics, issues, and subsequent solutions to an MDAS were discussed. Many of the issues in an MDAS can be addressed through work previously done by mobile system and multidatabase system researchers.

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 address important issues in the face of increased throughput and the limitations imposed by technology. A multidatabase transaction manager must process global transactions between loosely coupled, heterogeneous systems, 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 has been presented and evaluated in this chapter. The advantage of this algorithm is that the global performance of the system is increased by dynamically adjusting the amount of communication required to detect and resolve conflicts. A simulator was developed in C++ using CSIM in order to evaluate the performance. The results 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. Furthermore, it was shown how the proposed algorithm is used to decrease the communication requirements, resulting in higher global performance.

7.2 Future Directions

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

- The impact of the Internet on data management is growing at a tremendous pace. Considerations of how this data (both structured and unstructured) can and should be integrated into MDAS systems is important.
- Support is needed to overcome disconnection and weak connection. Lower bandwidth and disconnection are common in an MDAS environment; local caches, prefetching, queueing, and data broadcasting have been suggested as ways of alleviating these restrictions. However, due to the local autonomy restrictions in the MDAS environment, a better method/protocol should be investigated to address this issue.
- 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 sub-transactions were allowed to dominate the local systems, the overall global throughput would increase, while the throughput of local systems may decrease. It should be determine 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 that they 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. Also, the impact of non-uniform communication requirements between the client and server as well as between servers should be investigated.

Appendix: Related Projects

There are several ongoing and completed projects which address the issues involved with mobile computing. Tables IX and X summarize the different mobile projects. These include various research/experimental projects along with a brief summary of their characteristics.

7.3 Mobile-Transparent Projects

Mobile-transparent systems include the Coda project [21, 34, 35], the Ficus project [40], the Little Work project [22], and the BNU project [50], which are summarized in Table IX. These projects all offer file system support for mobile clients.

7.3.1 Coda

Coda, developed at Carnegie Mellon University (CMU) as a descendant of the Andrew File System (AFS) [34], pioneered many of the distributed services for mobile clients. In particular, a cache and an operation log provides support for weak connectivity and disconnect operations [35]. Coda logs all transactions during periods of disconnection and replays the log upon reconnection. The cache manager, called *Venus*, is present in each client, and operates in one of three states: (1) hoarding, (2) emulating, and (3) reintegrating. The hoarding state is the "normal" state while connected to the server. The primary function of this state is to ensure that frequently used objects are cached during a connection [34]. When a disconnection occurs, the cache manager enters the emulating state and services all file system requests from the cache contents. If a cache miss occurs, it appears as a failure to the user. Upon reconnection, Venus resynchronizes the cache

TABLE IX
SUMMARY OF MOBILE-TRANSPARENT SYSTEMS

Project name	Location	Status	Brief summary of features	Cache/prefetching
BNU [50]	University of Washington	Research/ Experimental	RCP based proxy system providing access to data.	No/No
CODA [21, 34, 45]	Carnegie Mellon University	Experimental	Distributed file system that pioneered support for weak connectivity and the disconnect operation through the use of a cache, operation log, and rapid cache validation.	Yes/Yes
Ficus [40]	UCLA	Experimental	Distributed file system using replication and an optimistic concurrency control policy.	No/No
Little Work [22]	U. Michigan, Ann Arbor	Experimental	Cache based distributed file system.	Yes/Yes

TABLE X
SUMMARY OF MOBILE-AWARE SYSTEMS

Project name	Location	Status	Brief summary of features	Cache/Prefetching
Bayou [15]	Xerox PARC	Research/ Experimental	Data-sharing application using replication. Supports session guarantees and user selectable placement use of data.	Yes/Yes
Daedalus [47]	U.C. Berkeley	Research/ Experimental	Proxy-based system which distils data, transmitting only enough information which can be sent over a low bandwidth link.	No/No
GloMop [28]	U.C. Berkeley	Research/ Experimental	Closely related to the Daedalus project, this system primary focus is the connection between the proxy and the client.	No/No
InfoPad [17, 28, 36]	U.C. Berkeley	Research/ Experimental	Portable dumb-terminal capable of displaying text, video and graphics. All functionality is provided by computing nodes on a fixed network.	No/No
Rover [23, 24, 25]	M.I.T.	Research/ Experimental	RCP based toolkit which supports relocatable dynamic objects and queued remote procedure calls.	Yes/Yes
Wit [51]	University of Washington	Research/ Experimental	API based object-oriented proxy system.	Yes/Yes

with servers in the reintegrating state and returns to the hoarding state upon completion.

The use of volume validation (called rapid cache validation) and asynchronous data synchronization techniques are used to deal with weak connections. The resynchronization of the cache with weak connectivity may take a substantial amount of time. Rapid cache validation involves the use of version stamps for each volume and individual object on a server, which are incremented when objects are updated. Clients then store and use this information when reintegrating the cache. If volume stamp is still valid, then every object in the volume is valid, and the reintegration is complete. When a volume stamp is determined to be invalid, each object is validated individually. Finally, updates to the server are asynchronously propagated to the server in the background. Records must spend a minimal amount of time (called an aging window) in the operational log before reintegration is performed.

7.3.2 Ficus

Ficus is a research project at the University of California, Los Angeles (UCLA), which is a distributed file system that uses replication and optimistic concurrency control [40]. The system allows any replica of a file to be updated at any time, allowing for a very high degree of availability. Furthermore, replication of data to the local site is used to support disconnection. The tradeoff, however, is that conflicts result from this policy. Conflicts include: update/update conflicts, name conflicts, and remove/update conflicts. Ficus guarantees that conflicting updates are detected by using a version vector [40]. Each file replica has its own version vector, which keeps track of the history of updates to a file. Version vectors are compared in order to detect conflicts. When conflicts are detected, the file is marked as "in conflict," and normal operations to the file are suspended until the conflict is resolved. In some environments, up to two-thirds of the conflicts could be resolved automatically by Ficus without user intervention [40].

7.3.3 Little Work

Whereas the Ficus project uses replication to support the disconnect operation, the Little Work project is a file-based system which uses a cache [22]. The project is an experimental system from the University of Michigan, Ann Arbor. The cache manager services all requests whether connected or disconnected. While disconnected, if an operation requires the server, the cache manager logs the request. An optimistic control algorithm is used to detect conflicts. Upon reconnection, if no conflicts are detected, then a reconnect manager submits the operation to the server. If a conflict is detected, the user is notified. The Little Work project is similar to Coda, except that Little Work does not modify the servers in any way.

7.3.4 BNU

The BNU project uses proxy processes running on a fixed network which provide access to data in order to address mobility issues of handheld and limited capability processing devices [50]. The proxy processes run on workstations in the LAN. A remote procedure call (RPC) system provides the communication between a mobile system and the proxies. The proxies provide the transparency between the mobile system and application. A single RPC call can perform multiple operations, thus requiring less bandwidth. Furthermore, by using RPCs, the ability to load certain routines and data on the small device can be used to mask a disconnection. Non-blocking RPCs are queued by the proxy and immediately acknowledged, reducing latency and allowing for the delivery of the RPC to occur in parallel with application processing. Furthermore, queuing allows for multiple RPCs to the proxy to be unified into a single, equivalent RPC to the hand-held device.

7.4 Mobile-Aware Projects

Mobile-aware projects include the Rover project [23, 24, 25], Bayou project [15], the Wit project [51], the InfoPad [17, 28, 36] and the Daedalus/GloMop [28, 47]. These projects are summarized in Table X.

7.4.1 Rover Toolkit

The Rover project is a toolkit from the Massachusetts Institute of Technology (MIT) that supports relocatable dynamic objects and queued remote procedure calls (QRPC) [23]. The applications developed with this toolkit are object-based distributed client/server architecture. This system supports the development of mobile-aware as well as mobile-transparent applications.

A relocatable dynamic object is a well-defined object and interface that can be dynamically loaded to clients or servers. All application code and data are written as RDOs, and each RDO has a "home" server which maintains the primary copy. RDOs can be replicated and stored/prefetched in a client cache in order to improve bandwidth tolerance and provide a means to work during a disconnection. Furthermore, the relocatability of an RDO gives an application control over the location for computation, and is

particularly useful for exporting computation to servers or for distilling data. RDOs can also use compression and decompression in order to reduce network and storage utilization.

A QRPC is a non-blocking RPC that allows operations to be performed while the client is either connected or disconnected. Clients fetch RDOs from servers with QRPCs. Subsequently, the QRPC is stored in a local stable log. While the client is connected to the network, the Rover scheduler forwards any QRPCs present in the log to the server. Cached copies of an RDO are lazily propagated to the server when there is sufficient bandwidth. QRPCs use split-phase communication, where the request and response pair of the communication is split. This allows the client to use separate channels for transmitting and receiving data, allowing the most efficient, available channel to be used while allowing a client to power down while waiting for a pending operation.

The Rover toolkit was used to implement both mobile-aware and mobile-transparent applications. Experimental results showed that for mobile-transparent applications improvements of up to 17%, while performance improvements for mobile-aware applications were shown to be up to 750%.

7.4.2 Wit

The Wit project is based upon the BNU project and is implemented in the form of an application-programming interface (API). The Wit designers define application partitioning as the assignment of functionality to both mobile and stationary devices through partitioning data and functions into hyperobjects. Hyperobjects are linked hierarchical objects, which are managed by the system and are migrated or replicated across the wireless connection. Links are directional and represent some type of relationship between hyperobjects. Data semantics (in the form of hyperobjects and links) and observation of access patterns are used in order to make informed policy decisions about resource allocation and usage [51].

Caching attributes of an object are specified by the application. Normally, an object is replicated on the mobile device, and the cache consistency is maintained by explicit synchronization. Prefetching is also performed based upon access patterns and the semantics contained in the hyperobject links. Finally, a form of data distillation is performed on the retrieved data in order to reduce the overall data which must be transferred. The system has only been proposed and has not been implemented.

7.4.3 Bayou

The Bayou project is from Xerox PARC and is designed to support data

sharing among mobile users [15]. The design is a flexible client-server architecture. The architecture is designed such that "lightweight" servers will reside on portable machines. Similar to the Ficus project, the Bayou project uses replication to provide a high degree of availability of data, even while disconnected, to users. The policy used is a read-any/write-any replication scheme, where the user is able to read/write to any copy of the replicated data, and the data is only weakly consistent. Consistency is maintained by using a peer-to-peer anti-entropy policy for propagation of updates [15]. Anti-entropy ensures that all copies of the data are all converging towards the same state, where each server receives all write operations performed and correctly order the operations. And similar to Ficus, Bayou guarantees that conflicting updates are detected by using a version vector [15].

The system provides a client with a view of the replicated data, which is consistent with its own actions, through session guarantees. A session is an abstraction for the sequences of read and write operations performed on the data. Four types of guarantees are provided on a per-session basis to the user [15]:

- Read Your Writes—read operations reflect previous writes;
- Monotonic Reads—successive reads reflect a non-decreasing set of writes;
- Writes Follow Reads—writes are propagated after reads upon which they depend;
- Monotonic Writes—writes are propagated after writes that logically precede them.

7.4.4 Odyssey

Odyssey is a follow-up project to Coda that adds application-aware adaptation to the system. This system provides a fixed number of representations of data objects on a server, allowing for end-to-end bandwidth management. An API is provided for a client to track its current "environment" and negotiate for a representation that is appropriate for the current connectivity. It uses the concept of data fidelity, resource negotiation, and dynamic sets in order to effectively use and control system resources [45].

7.4.5 Daedalus/GloMop/Infopad

The Daedalus/GloMop project is a mobile computing project from the University of California, Berkeley. It runs processes on well-connected

workstations that act as a proxy for a mobile client that is on the other side of a low-bandwidth connection. The proxy then distills and/or refines the data, transmitting only enough information to the client in a format that can be transmitted over the low-bandwidth link. The idea is to trade processing cycles for bandwidth. Datatype-specific distillation is a highly lossy, datatype-specific compression that preserves most of the semantic content of a data object while adhering to a particular set of constraints [47]. Three datatypes were mainly investigated: images, formatted text, and video streams. Datatype-specific refinement is the process of fetching some part of a source object at increased quality. The distillation and refinement occur on-demand in order to provide the best possible results to applications in a changing mobile environment.

The InfoPad [17, 28, 36] project uses a portable dumb-terminal, which is able to display text, video, and graphics. Computing nodes on a fixed network provide all of the functionality; the terminal (Pad) only displays the information. The Pad differs from other access devices in that it does not contain a general purpose processor. Applications and user-interfaces are currently under development, which use only pen and audio as inputs, allowing them to operate without a keyboard.

Glossary

ACID: (Atomicity, Consistency, Isolation, Durability). Four properties used to define the correctness of a transaction.

ATM: (Asynchronous transfer mode). A 53-byte, fixed communication protocol commonly used in high-speed networks.

Autonomy: Autonomy refers to the ability of a system to operate independently, without the help or resources of another system. There are three different forms of autonomy: design, communication, and execution.

Data distillation: Data distillation is a process where incoming data is processed such that only portions of the data are displayed. Data distillation may be performed in order to accommodate display restrictions, or to reduce bandwidth requirements of a system.

DBMS: (Database management system). An information storage and retrieval system.

HRAD: (Heterogeneous remote access to data).

ISP: (Internet service provider). An ISP provides internet access to a user. Typically the service is available though a modem connection.

LAN: (Local area network). A group of computers connected together which share resources.

Long-lived transaction: A transaction which is in an open or non-committed state for a long duration of time. These types transactions may last from several hours to days, and even weeks.

MDAS: (Mobile data access system). A term used to describe a new computational environment in which a wireless mobile computing environment is superimposed on a multidatabase environment providing efficient access heterogeneous data sources.

MDBMS: (Multidatabase management system). A global system layer that allows distributed access to multiple preexisting databases.

Mobile awareness: A system or application is alert to changes in the condition of its environment, and is able to adapt to those changes.

Mobility: The ability to physically move a computing device and use it in a different location or while actually moving.

NC: (Network computer). A low cost, low maintenance computer system connected to a network. These systems are usually diskless and rely heavily upon a server.

PDA: (Personal digital assistant). A hand held computing device.

Remote access: A fixed or mobile node that accesses data over a network connection characterized by lower bandwidth, frequent disconnection, and higher error rates.

SSM: (Summary schemas model). An adjunct to a multidatabase system used for query resolution.

References

[1] Agrawal, R., Carey, M., and Livny, M. (1985). Models for studying concurrency control performance: alternatives and implications. *Proceedings of the ACM SIGMOD International Conference on Management of Data.*

[2] Alonso, R., Garcia-Molina, H., and Salem, K. (1987). Concurrency control and recovery for global procedures in federated database systems. *Data Engineering*, **10**(3), 5–11.

[3] Badrinath, B. R. (1996). Designing distributed algorithms for mobile computing networks. *Computer Communications*, **19**(4).

[4] Baker, M. G. (1994). Changing communication environments in MosquitoNet. *Proceedings of the 1994 IEEE Workshop on Mobile Computing Systems and Applications*, December.

[5] Bernstein, P., Hadzilacos, V., and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*, Addison-Wesley Publishing Company.

[6] Breitbart, Y., Silberschatz, A., and Thompson, G. (1987). An update mechanism for multidatabase systems. *IEEE Data Engineering Bulletin*, **10**(3), 12–18.

[7] Breitbart, Y., Georgakopoulos, D., Rusinkiewicz, M., and Silberschatz, A. (1991). On rigorous transaction scheduling. *IEEE Transactions on Software Engineering*, **17**(9).

[8] Breitbart, Y., Garcia-Molina, H., and Silberschatz, A. (1992). Overview of multidatabase transaction management. *The VLDB Journal*, **1**(2), 181–239.

[9] Breitbart, Y., and Silberschatz, A. (1993). Performance evaluation of two multidatabase transaction management algorithms. *Computing Systems*, **6**(3).

[10] Bright, M., Hurson, A., and Pakzad, S. (1992). A taxonomy and current issues in multidatabase systems. *IEEE Computer*, **25**(3), 50–60.
[11] Bright, M. W., Hurson, A. R., and Pakzad, S. H. (1994). Automated resolution of semantic heterogeneity in multidatabases. *ACM Transactions on Database Systems*, **19**(2).
[12] Conover, J. (1997). Solutions for an ATM upgrade. *Network Computing*, 15 May.
[13] Cormen, T., Leiserson, C., and Rivest, R. (1990). *Introduction to Algorithms*, McGraw-Hill Book Company.
[14] Dash, K., Hurson, A., Phoha, S., and Chehadeh, C. (1994). Summary schemas model: a scheme for handling global information sharing. *Proceedings of the International Conference on Intelligent Information Management Systems*, pp. 47–51.
[15] Demers, A., Pertersen, K., Spreitzer, M., Terry, D., Theier, M., and Welch, B. (1994). The Bayou architecture: support for data sharing among mobile users. *IEEE Proceedings of the Workshop on Mobile Computing Systems and Applications*, December.
[16] Elmagarmid, A., Jing, J., and Furukawa, T. (1995). Wireless client/server computing for personal information services and applications. *ACM Sigmod Record*.
[17] Fox, A., Gribble, S. D., Brewer, E. A., and Amir, E. (1996). Adapting to network and client variability via on-demand dynamic distillation. *Proceedings of ASPLOS-VII*, Boston, MA, October.
[18] Garcia-Molina, H. (1988). Node autonomy in distributed systems. *IEEE Proceedings of the International Symposium on Databases and in Parallel and Distributed Systems*.
[19] Geogakopoulos, D., Rusinkiewicz, M., and Sheth, A. (1994). Using tickets to enforce the serializability of multidatabase transactions. *IEEE Transactions on Knowledge and Data Engineering*, **6**(1), 166–180, February.
[20] Gray, J., and Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann.
[21] Hills, A., and Johnson, D. B. (1996). Wireless data network infrastructure at Carnegie Mellon University. *IEEE Personal Communications*, **3**(1), February.
[22] Honeyman, P., Huston, L., Rees, J., and Bachmann, D. (1992). The LITTLE WORK Project. *Proceedings of the Third IEEE Workshop on Workstation Operating Systems*, April.
[23] Joseph, A. D., Tauber, J. A., and Kaashoek, M. F. (1997). Mobile computing with the Rover toolkit. *IEEE Transactions on Computers: Special Issue on Mobile Computing*, February.
[24] Joseph, A. D., and Kaashoek, M. F. (1996). Building reliable mobile-aware applications using the Rover toolkit. *Proceedings of the 2nd ACM International Conference on Mobile Computing and Networking*, November.
[25] Joseph, A. D., deLespinasse, A. F., Tauber, J. A., Gifford, D. K., and Kaashoek, M. F. (1995). Rover: A toolkit for mobile information access. *Proceedings of the Fifteenth Symposium on Operating Systems Principles*, December.
[26] Kaashoek, M. F., Pinckney, T., and Tauber, J. A. (1995). Dynamic documents: mobile wireless access to the WWW. *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, December.
[27] Lai, S. J., Zaslavsky, A. Z., Martin, G. P., and Yeo, L. H. (1995). Cost efficient adaptive protocol with buffering for advanced mobile database applications. *Proceedings of the Fourth International Conference on Database Systems for Advanced Applications*, April.
[28] Le, M. T., Burghardt, F., Seshan, S., and Rabaey, J. (1995). InfoNet: The networking infrastructure of InfoPad. *Proceedings of Compcon*, March.

[29] Lim, J. B., Hurson, A. R., and Chehadeh, C. (1997). *Heterogeneous Data Access in a Mobile Environment—Issues and Solutions*. Department of Computer Science and Engineering Technical Report CSE-97-010, September.
[30] Mayer, J. (1995). Wireless stretches LANs. *Computer Design*, February.
[31] Mehrotra, S., Korth, H., and Silberschatz, A. (1997). Concurrency control in hierarchical multidatabase systems. *The VLDB Journal*, **6**, 152–172.
[32] Motorola, ReFLEX Fact Sheet, 1997.
[33] Noble, B. B., and Helal, A. (1995). Mobile computing and databases: anything new? *Sigmod Record*, December.
[34] Noble, B. D., and Satyanarayanan, M. (1994). An empirical study of a highly available file system. *Proceedings of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*.
[35] Ortiz, R., and Dadam, P. (1995). Towards the boundary of concurrency. *2nd International Conference on Concurrent Engineering, Research and Applications*.
[36] Padmanabhan, V. N., and Mogul, J. C. (1996). Using predictive prefetching to improve World Wide Web latency. *ACM SIGCOMM Computer Communication review*, July.
[37] Perterson, T., and Yegyazarian, A. (1998). The New Networked Computers. *PC Magazine*, pp. 211–229, May.
[38] Pitoura, E., and Bhargava, B. (1995). Maintaining consistency of data in mobile distributed environments. *Proceedings of 15th International Conference on Distributed Computing Systems*, pp. 404–413.
[39] Prakash, R., and Singhal, M. (1996). Low-cost checkpointing and failure recovery in mobile computing systems. *IEEE Transactions on Parallel and Distributed Systems*, **7**(10), October.
[40] Reiher, P., Heidemann, J., Ratner, D., Skinner, G., and Popek, G. (1994). Resolving file conflicts in the Ficus file system. *Proceedings of the 1994 USENIX Conference*.
[41] Rogers, A. (1997). Networks in Space. *Communications Week*, February 17.
[42] Rogers, A. (1997). 56K on the way. *Communications Week*, February 3.
[43] Satyanarayanan, M. (1996). Mobile information access. *IEEE Personal Communications*, **3**(1), February.
[44] Satyanarayanan, M. (1996). Fundamental challenges in mobile computing. *15th ACM Symposium on Principles of Distributed Computing*, May.
[45] Satyanarayanan, M., Noble, B., Kumar, P., and Price, M. (1994). Application-Aware Adaptation for Mobile Computing, *Proceedings of the 6th ACM SIGOPS European Workshop*, September.
[46] Satyanarayanan, M., Kistler, J. J., and Mummert, L. B. (1993). Experience with disconnected operation in a mobile computing environment. *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*, August.
[47] Seshan, S., Balakrishnan, H., and Katz, R. H. (1996). Handoffs in cellular wireless networks: The Daedalus implementation and experience. *Kluwer International Journal on Wireless Communications*.
[48] Sohn, K., and Moon, S. (1997). Achieving a high degree of concurrency in multidatabase transaction scheduling. *Proceedings of the 5th International Conference on Database Systems for Advanced Applications*, April 1–4.
[49] Stallings, W. (1994). *Data and Computer Communications*, 4th Edn, Macmillan Publishing Company, Englewood Cliffs, NJ.
[50] Watson, T., and Bershad, B. N. (1993). Local area mobile computing on stock hardware and mostly stock software. *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*.

[51] Watson, T., Effective wireless communication through application partitioning. *Proceedings of the 5th Workshop on Hot Topics in Operating Systems.*
[52] Wolski, A., and Veijalainen, J. (1990). 2PC agent method: Achieving serializability in presence of failures in a heterogeneous multidatabase. *Proceedings of the International Conference on Databases, Parallel Architectures and Their Applications*, pp. 321–330.
[53] Zdonik, S., Alonso, R., Franklin, M., and Acharya, S. (1994). Are disks in the air just pie in the sky? *Proceedings of Workshop on Mobile Computing Systems and Applications*, pp. 1–8.
[54] Zhang, A., Bhargava, M., and Bukhres, O. (1994). Ensuring relaxed atomicity for flexible transactions in multidatabase systems, *Proceedings of ACM-SIGMOD International Conference on Management of Data.*

The World Wide Web

HAL BERGHEL AND DOUGLAS BLANK

Department of Computer Science
University of Arkansas
Fayetteville, AR 72701
USA

Abstract

This article provides a high-level overview of the World Wide Web in the context of a wide range of other Internet information access and delivery services. This overview will include client-side, server-side and "user-side" perspectives. Underlying Web technologies as well as current technology extensions to the Web will also be covered. Social implications of Web technology will also be addressed.

ADVANCES IN COMPUTERS, VOL. 48
ISBN 0-12-012148-4