Mobile Networks and Applications 5 (2000) 311–321

A pre-serialization transaction management technique for mobile multidatabases

Ravi A. Dirckze and Le Gruenwald

School of Computer Science, University of Oklahoma, Norman, OK 73019, USA

Rapid advances in hardware and wireless communication technology have made the concept of mobile computing a reality. Thus, evolving database technology needs to address the requirements of the future mobile user. The frequent disconnection and migration of the mobile user violate underlying presumptions about connectivity that exist in wired database systems and introduce new issues that affect transaction management. In this paper, we present the Pre-Serialization (PS) transaction management technique for the mobile multidatabase environment. This technique addresses disconnection and migration and enforces a range of atomicity and isolation criteria. We also develop an analytical model to compare the performance of the PS technique to that of the Kangaroo model.

1. Introduction

A Multidatabase System (MDBS) is a federation of pre-existing database systems, and is the natural result of shifting priorities and the need of an organization to be part of a greater information system [7]. In the Asilomar report [1], the authors state that "in the future, billions of web clients will be accessing millions of databases, and that the world wide web will be one large federated system". The rapid advances in wireless communication technology dictates that these static database systems extend their services to the mobile user.

Transaction management in the Mobile Multidatabase (MMDB) environment has been the focus of many publications [3,6,12,14]. However, existing techniques do not address two key issues. First, the techniques do not address the Isolation property of global transactions. Second, they fail to address disconnection that represents catastrophic failures. In this paper, we propose a Pre-Serialization (PS) transaction management technique for the MMDB environment that addresses the deficiencies that exist in the current literature. We develop an analytical model of a Mobile Multidatabase System (MMDBS) that is used to evaluate the performance of the PS techniques and to compare its performance to that of the Kangaroo model [6]. The Kangaroo model is chosen as it is the most recent of the existing techniques and is the only model to capture the movement behavior of the user.

This paper is organized as follows. Section 2 provides a brief introduction to the MMDB environment and a discussion on the responsibilities of the transaction management process. The PS techniques is presented in section 3. The literature review is presented in section 4 followed by the performance evaluation study in section 5. Concluding remarks are presented in section 6.

2. MMDB transaction management

2.1. MMDB architecture

The mobile computing model consists of two distinct sets of entities: a fixed network system and a continuously changing set of mobile clients (figure 1). Some units on the static network have the capability of communicating with mobile units through some wireless medium and are called Mobile Support Stations (MSS). The wireless communication medium includes cellular architecture, satellite services, wireless LAN, etc. The typical mobile hosts are portable computers that have limited resources compared to their desktop counterparts [12]. Due to the unreliability of the wireless communication medium as well as limited resources available, the mobile user will be characterized by frequent disconnection. However, this disconnection is predictable [11].

The MMDBS consists of a set of autonomous databases and a set of software modules residing on the fixed network that are collectively referred to as the Mobile Multidatabase Management System (MMDBMS) (figure 2). The respective local database systems (LDBSs) retain complete control over their databases. Each LDBS provides a service

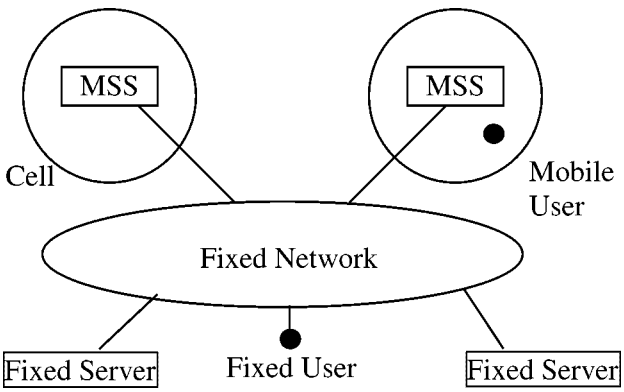


Figure 1. The mobile computing architecture.

© Baltzer Science Publishers BV

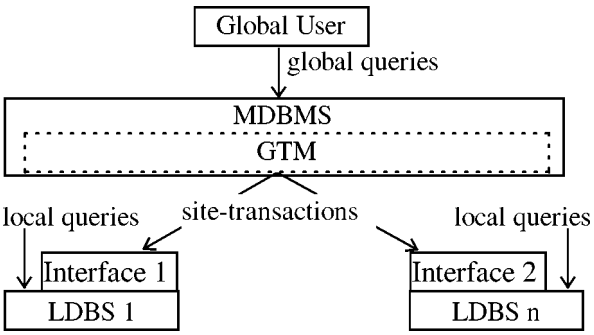


Figure 2. The mobile multidatabase system.

interface that specifies the operations accepted from and the services provided to the MMDBMS. The MMDBMS cooperates with the service interfaces to provide extended transaction services to users. All local transaction processing remains transparent to the MMDBMS. Global users – users connected to the MMDBMS – are capable of accessing multiple databases by submitting global transactions to the MMDBMS. Global users can be either static users with fixed connections to the network or mobile users.

A global transaction consists of a set of operations, each of which is a legal operation accepted by some service interface. Any subset of operations of a global transaction that access the same site may be executed as a single local transaction and will form a logical unit called a site-transaction. As mobile users migrate from one MSS to another, operations of global transactions may be submitted from different MSSs. Such transactions are referred to as migrating transactions.

2.2. Responsibilities of the GTM

The Global Transaction Manager (GTM) is responsible for providing consistent and reliable units of computing, i.e., (transactions) to the data within its domain. Generally, this can be achieved by enforcing the ACID (Atomicity, Consistency, Isolation, and Durability) properties [9]. However, the applicability of ACID in the MMDB environment has been questioned. First, as a consequence of autonomy, we can assume that there are no integrity constraints defined across different LDBSs [5]. As each LDBS will ensure that site-transactions do not violate any local integrity constraints, global transactions will, by default, satisfy the global consistency property. Similarly, the GTM can rely on the durability property of the local LDBS to ensure durability of committed global transactions. Thus, the GTM needs only enforce the Atomicity and Isolation (A/I) properties. Second, in [6], the authors make a compelling argument for providing unrestricted access to data in the MMDB environment: "Returning dirty data tagged with appropriate warnings is much more useful than returning an ABORT message . . . ". Thus, the GTM needs to support a spectrum of A/I correctness criterion.

In addition to the A/I properties, the GTM needs to address disconnection and migrating transactions. Unlike in the static environment, disconnection in the mobile environment cannot always be treated as failures that result in aborted transactions. In some cases, however, disconnection will be caused by a catastrophic failure. Halted transactions will not be resumed after a catastrophic failure. As the MMDBMS can only predict catastrophic failures, aborting disconnected transactions is likely to result in some untimely terminations. The GTM needs to take appropriate steps to minimize such untimely terminations.

Disconnection and migration of the mobile user prolong the execution time of mobile transactions, which consequently affects the Isolation property [4]. In order to maintain a notion of fairness, the concurrency control mechanism of the GTM must not penalize mobile transactions for this prolonged execution. In addition, to tolerate long-lived transactions (LLT) without disruption to local transaction processing, site-transactions must be allowed to commit early so that (local) resources may be released in a timely fashion.

Furthermore, the GTM must also conform to multidatabase design restrictions, i.e., the autonomy of the LDBSs cannot be violated.

3. A pre-serialization transaction management technique

In this section, we present the Pre-Serialization (PS) transaction management technique for the MMDB environment. This technique allows LLTs to establish their serialization order prior to completing their execution. It supports disconnection and migration and addresses catastrophic failures that may occur. We also, introduce a PGSG algorithm that enforces a range of A/I correctness criteria.

3.1. The model

Global transactions are based on the multi-level transaction model [9] in which the global transaction consists of a set of compensatable sub-transactions. A compensatable transaction is a transaction whose effects can be undone after it has committed by executing a compensating transaction [10]. In the proposed model, all operations of a global transaction that access the same site constitute a single site-transaction (analogous to a subtransaction) and is executed as a single (local) transaction. Global transactions are limited to no more than one site-transaction per site as site-transactions are executed as independent transactions by the LDBS and the MMDBS cannot prevent the LDBS from executing local transactions between site-transactions (and exposing different states of the database to the global transaction). As all site-transactions are compensatable, they are committed at the LDBS prior to the decision to commit the global transaction, thus releasing resources in a timely manner.

In addition, all site-transactions will be categorized as either vital or non-vital [3]. All vital site-transactions of

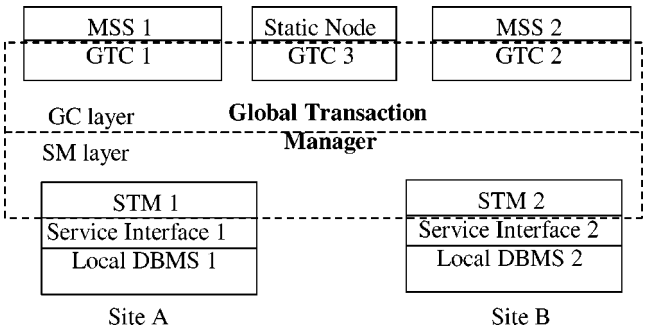


Figure 3. Global Transaction Manager.

a global transaction must succeed in order for the global transaction to succeed. The abort of a non-vital site-transaction does not force the global transaction to be aborted. The A/I properties are enforced only on the set of vital site-transactions. This gives the PS technique the flexibility to accommodate the full range of A/I correctness criteria as follows: If all site-transactions of a global transaction are classified as vital, then strict A/I will be enforced. On the other hand, if all site-transactions are classified as non-vital, then the A/I properties are not enforced. The time between the submission of the first vital site-transaction and the completion of the last vital site-transaction is called the vital phase of a global transaction. For simplicity, global transactions of mobile users are classified as LLTs and global transactions of static users are called non-LLTs.

3.2. The Global Transaction Manager

The GTM consists of two layers: a Global Coordinator (GC) layer and a Site Manager (SM) layer (figure 3). The GC layer consists of a set of Global Transaction Coordinators (GTCs) such that there exists a GTC at each MSS and any other static node that supports external users. Global transactions are initiated at some GTC which will submit site-transactions to the STMs, handle disconnection and migration of the mobile user, log responses that cannot be delivered to the disconnected user, enforce the atomicity and isolation properties, etc. The SM layer consists of a set of Site Transaction Managers (STMs) such that there exists an STM at each participating LDBS. Each global transaction can be in one of five states:

(1) active – the user is connected and execution continues,

(2) disconnected – the user is disconnected, but the disconnection was predicted and re-connection is expected,

(3) suspended – the user is disconnected and is deemed to have encountered a catastrophic failure,

(4) committed – the transaction committed successfully, and

(5) aborted – the transaction is aborted.

Note that disconnected and suspended states do not apply to global transactions of static users. Each global transaction is associated with a global (data) structure (table 1) that is maintained by the associated GTC. This global structure migrates with the mobile user transaction from GTC to GTC.

Table 1
Global data structure.

GTID	global transaction identifier
GTType	mobile (LLT) or static (non-LLT)
GTStatus	current state of global transaction
IsolationVerified	isolation verified (yes/no)
SiteList	respective site of each site-trans.
STIDList	STID of each site-transaction
CriticalList	vital/non-vital for each site-trans.
STIDStatusList	status of each site-transaction
ResponseList	list of undelivered responses

Table 2
Site table structure.

GTID	respective GTID
STID	assigned STID
MSS_ID	current MSS of user
STIDStatus	current state of site-transaction
CompTransaction	compensating site-transaction

The STM at each site supervises the execution of site-transactions submitted to that site. Each site-transaction can be in one of four states:

(1) active – the site-transaction is active,

(2) completed – the site-transaction has committed at the local database but the global transaction has not committed,

(3) aborted – the site-transaction is aborted, or

(4) committed – the site-transaction and the respective global transaction have committed.

Each STM will maintain a site table containing information on all site-transactions submitted to it (table 2).

The GTC submits all site-transactions and their compensating (site) transactions to the respective STMs. Upon completion of each site-transaction, the STM will submit a commit operation to the LDBS and update the STIDStatus to reflect the outcome of the local commit operation, i.e., marked completed or aborted. The outcome will then be conveyed to the GTC to be recorded in the global structure.

Whenever a user disconnects, the respective GTStatus is marked as disconnected. The execution of disconnected transactions is not halted. Upon reconnection, the GTStatus of disconnected transactions will be set to active and execution proceeds. All responses received after disconnection are placed in the ResponseList and delivered to the user upon re-connection. At any time during a period of disconnection, if the MMDBS determines that a catastrophic failure has occurred, the respective GTStatus is marked as suspended and the execution is halted, i.e., no new site-transactions are initiated. In order to minimize erroneous aborts, suspended global transactions are not aborted until they obstruct the execution of other global transactions.

At the end of execution of a non-LLT, the GTC executes the PGSG algorithm to verify the A/I properties. If A/I have not been violated, the transaction is committed; else, it is aborted. In the case of an LLT, the GTC executes the PGSG algorithm at the end of its vital phase. If the A/I properties have not been violated, its serialization order is registered in the global serialization scheme and the transaction is toggled (i.e., isolation verified is set to true). Thus, an LLT that has been toggled is guaranteed not to be aborted due to concurrency conflicts unless it obstructs the execution of another global transaction while in a suspended state. As LLTs are allowed to establish their serialization order prior to completing their execution, the prejudicial treatment of mobile global transactions is minimized. An LLT is allowed to continue to initiate non-vital site-transactions after being toggled and is committed at the end of its execution.

This paper introduces a Partial Global Serialization Graph (PGSG) algorithm that verifies the serializability of a global transaction by constructing only a "partial" global serialization graph from the local serialization information collected by each STM. This algorithm (which is executed by the GTC) is able to determine the subset of local serialization graphs that need to be investigated in order to verify the serializability of a global transaction. The key to the algorithm is propagation – global serialization information that is distributed to selected STMs during its execution.

3.3. The PGSG algorithm

The atomicity property of PGSG is based on *Semantic Atomicity* [10]. Semantic Atomicity is satisfied if either, all site-transactions are committed, or each site-transaction is aborted or compensated for. The isolation property is based on serializability. In the multidatabase environment, serializability requires that any two global transactions that conflict be serialized in the same order at all sites at which they conflict. The STM at each LDBS maintains a Site Serialization Graph (SSG) which is used by the PGSG algorithm to verify the isolation property.

As the serialization order of site-transactions within the local databases is transparent to the STM, this information is obtained implicitly by forcing conflicts among the vital site-transactions using a variation of the ticket method proposed in [8]. In addition to the operations (queries) accepted by the site, the service interface specifies which operations potentially conflict with each other. For each set of conflicting operations, the LDBS maintains a different ticket. At the beginning of its execution, each vital site-transaction reads the respective tickets (of all operations executed by it), increments each ticket and writes the new value back to the database. Each ticket value read by the vital site-transaction indicates its (local) serialization order with respect to all other vital site-transactions that execute conflicting operations [2]. Note that, any site-transactions that violate the local Isolation property will be aborted by the local LDBS. The ticket values read by each vital site-transaction will be used to determine its serialization order with respect to other vital site-transactions that execute at that site.

Table 3
SSG node.

GTID	respective global transaction ID
GTStatus	status of global transaction
IsolationVerified	commit intent of global transaction
NodeCategory	accessed or propagated
SiteID	access or propagated SiteID

The SSG at each site is a directed graph whose nodes represent the GTIDs of the respective vital site-transaction and edges represent (forced) conflicts between their respective site-transactions executed at that site. For example, if $T_1 \rightarrow T_2$ exists in some SSG, then global transactions T_1 and T_2 access at least one common site where the ticket for some operation obtained by the site-transaction of T_1 is less than the ticket obtained by the site-transaction of T_2. The information contained within each node is given is table 3. Each node in the SSG is categorized as either an accessed node or a propagated node. An accessed node represents a global transaction that executed a site-transaction at that site. A propagated node represents a global transaction whose serialization order was copied to the SSG during the execution of the PGSG algorithm. Next, certain terms used in the algorithm are defined.

Definition 1. We say that T_j is *reachable* from T_i in graph G if there is a path from T_i to T_j in G, i.e., $T_i \rightarrow \cdots \rightarrow T_j$.

Definition 2. *Reachable*(T_j,) is a (directed) subgraph of an SSG that contains node T_j and all nodes T_i such that T_j is reachable from T_i in the SSG.

Definition 3. A *candidate* node is any propagated node whose GTStatus is not committed.

Definition 4. Let G_i and G_j be two graphs with node sets N_i and N_j and edge sets E_i and E_j, respectively. The operation $G = \textit{Merge}(G_i, G_j)$ results in a new graph $G(N, E)$ such that $N = N_i \cup N_j$ and $E = E_i \cup E_j$, where $\cup$ is the union operator.

Definition 5. The graph *Predecessor*(T_j, S_m) is the subgraph *Reachable*(T_j) of the SSG at site S_m merged with all *Predecessor*(T_k, S_n) graphs, where T_k is a candidate node in *Reachable*(T_j) and S_n is the respective propagated site of T_k. Formally, *Predecessor*(T_j, S_m) = {G = *Reachable*(T_j) | *Merge*(G, *Predecessor*(T_k, S_n)) $\forall$candidate nodes T_k in *Reachable*(T_j), where S_n is the SiteID of T_k}.

Definition 6. The list *PList*(T_j, S_m) is a list of sites maintained at site S_m that contains the list of sites from which the predecessor graphs were obtained in order to construct *Predecessor*(T_j, S_m).

The PGSG algorithm executed by the GTC is given below. The request argument specifies whether T_j is to be toggled or committed. First, this algorithm verifies the atomicity property. Next, it obtains the predecessor graphs from all primary sites of T_j. Primary sites are the sites at which T_j executed vital site-transactions. The STM at each primary site executes the Request Predecessor algorithm to construct the *Predecessor*(T_j) graph and submits it to the GTC executing the PGSG algorithm. The PGSG algorithm will then construct the Partial Global Serialization (PGS) graph, verify the isolation property, and either toggle commit or abort the global transaction. If the global transaction is to be committed or toggled, the PGS graph is sent to all participating sites so that the required serialization information is propagated.

PGSG algorithm (Request, T_j**)**
/* Verify A/I for global transaction T_j */
/* First, verify Atomicity */
If any critical site-transaction has been aborted
  Send Abort (T_j) to all sites in SiteList /* Abort all site-transactions */
Else
  /* verify Isolation */
for all site S_m in SiteList where T_j executed its vital site-transactions, obtain Predecessor(T_j, S_m)
  by executing the Request Predecessor(T_j, S_m) algorithm
Generate PGSG by Merging all Predecessor(T_j, S_m)
Check for cycles with respect to T_j, Committed nodes and Togged nodes
If cycles are detected
  If cycles can be broken by aborting Suspended global transactions
    Mark GTStatus of Suspended nodes as Aborted in PGSG
  Else /* Isolation violated */
    Send Abort (T_j) to all sites in SiteList /* Abort all site-transactions */
    Exit Algorithm
  End If
End If
/* A/I verified */
Mark IsolationVerified in Global Structure and node T_j in PGS graph as True
/* Propagate success and serialization information */
Send "Success" and PGS graph to sites in SiteList where T_j executed its vital site-transactions
End If
End {*PGSG Algorithm*}

The Request Predecessor module is executed at each Primary and Secondary site of T_j. Secondary sites are the sites deemed necessary to participate in constructing the PGSG graph for T_j. That is, some site already participating in the algorithm has an active propagated node containing this site as its SiteID. The secondary sites will submit the requested Predecessor graph to the Primary sites which will submit their Predecessor graphs to the GTC. Each site will wait for the outcome of the PGSG algorithm. If the T_j is to be committed, all participating sites will copy propagation information by merging *Reachable*(T_k) of the returned PGS graph where T_k is the transaction whose predecessor graph was requested, with *Reachable*(T_k) of its SSG. Any suspended transaction that is marked as aborted will be aborted by the STM.

Request Predecessor(T_j, S_m**)**
Construct Predecessor(T_j, S_m), PList(T_j, S_m)
Submit Predecessor(T_j, S_m) to requester
Wait for Reply from requestor
If Reply is Abort(T_j) /* site-transaction is to be aborted */
  If T_i is Accessed node in SSG /* this is a Primary site */
    Abort T_j if Active or compensate T_j if Completed
  End If
  Send Abort (T_j) to all sites in PList(T_j, S_m)
    /* inform all Secondary sites */
Else /* global transaction is to be toggled */
  If T_i is Accessed node in SSG /* this is a Primary site */
    Mark IsolationVerified as True
  End If
  /* propagate serialization information */
  SSG = Merge(SSG, Reachable(T_j) of received PGS graph)
  Abort all Suspended nodes marked as Aborted in PGS graph
  Send 'Success' and PGS graph to all sites in PList(T_j, S_m)
End If
End {*Request Predecessor*}

Example. In this example, the MMDBS consists of 3 sites labeled S_1 through S_3. There are 3 active global transactions labeled T_1 through T_3 in the system. For simplicity, we assume that each transaction accesses two sites, all site-transactions at each site conflict with each other, and that all transactions have completed their execution but have not

Table 4
Sample execution of PGSG algorithm.

	S_1	S_2	S_3	PGS graph
Initial SSGs	$T_3 \rightarrow T_1$	$T_1 \rightarrow T_2$	$T_2 \rightarrow T_3$	
Commit of T_1	$T_3 \rightarrow T_1$	$T_3 \rightarrow T_1 \rightarrow T_2$ [S_1]		$T_3 \rightarrow T_1$ [C]
Commit of T_3	$T_2 \rightarrow T_3 \rightarrow T_1$ [S_3]		$T_2 \rightarrow T_3$	$T_2 \rightarrow T_3$ [C]
Commit of T_2	$T_3 \rightarrow T_1$	$T_3 \rightarrow T_1$ [S_1]	T_3	$T_2 \rightarrow T_3 \rightarrow T_1 \rightarrow T_2$ [A]

yet committed. The algorithm is illustrated in table 4. The initial SSGs are given in row one. Initially, all nodes are accessed nodes. The next 3 rows reflects the SSGs after the completion of the PGSG algorithm of the specified transaction (only participating SSGs have new entries). Propagated nodes will contain their respective SiteIDs in brackets below the node. The last column reflects the PGS graph that is constructed at each stage. A [C] in the PGS graph states that the transaction is to be committed and an [A] states that it is to be aborted.

Assume that T_1 executes the PGSG algorithm first, T_3 second and T_2 third. For the commit of T_1, S_1 and S_2 participate as primary sites. (The predecessor graph submitted by S_1 is $T_3 \rightarrow T_1$, and the predecessor graph submitted by S_2 is T_1.) As there are no cycles in the PSG graph, T_1 commits successfully. After the commit, the PGS graph is sent to all participating sites (S_1 and S_2) for information propagation. Next, T_3 commits successfully (node T_2 is propagated to S_1). Finally, T_2 executes the PGSG algorithm. S_2 and S_3 participate as primary sites. As the SSG in S_2 has an active propagated node in its SSG with SiteID S_1 (i.e., T_3 which when propagated to S_2 was active and, therefore, still deemed to be active), S_1 will participate as a secondary site. As the PSG graph contains a cycle and T_2 is the last transaction in that cycle to attempt to commit, T_2 will be aborted.

3.3.1. Proof of correctness

Lemma 1. Let $T_i \rightarrow T_j$ be in the SSG at some site S_j. Then, T_i began its execution at S_j prior to the completion of T_j's execution at S_j and, therefore, prior to the (global) commit of S_j.

Proof. In order for $T_i \rightarrow T_j$ to exist, T_i must have obtained a ticket that is less than the ticket obtained by T_j. Therefore, T_i began its execution at S_j prior to T_j completing its execution at S_j. □

Theorem 1. Let $T = \{T_1, T_2, \ldots, T_n\}$ be a set of transactions that cause a cycle. Assume that T_2 through T_n commit successfully and that T_1 is the last transaction in T to attempt to commit. Then T_1 will be an accessed node as well as a candidate node in some *Predecessor*(T_1, S_x) used to construct the PGSG. Thus, the cycle will be detected.

Proof. For simplicity, let us assume that each transaction executes at exactly two sites such that the cycle

$$C \equiv T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n \rightarrow T_1 \text{ is produced.}$$

By lemma 1, for all T_q that have completed their execution, there exists $T_p \rightarrow T_q$ for some T_p in T in the SSG at some site S_q at which T_q executed. When T_q executes the PGSG commit algorithm, T_p is in *Predecessor*(T_q, S_q) used to construct the PGSG. Now, either T_p is committed, or not committed.

If T_p is not committed then T_p will be added as a candidate node to all the SSGs at which T_q executed.

If T_p is committed, then, by lemma 1, there exists a T_m in S such that $T_m \rightarrow T_p$ at some site S_m at which T_p executed. Once again, either T_m was committed or not committed at the time of T_p's commit. If T_m was not committed, then T_m was propagated to S_m at the time of T_p's commit and, as a result, will be in *Predecessor*(T_q, S_q) at the time of T_q's commit and will be added as a Candidate node to all SSGs at which T_q executed. If T_m was committed, we may repeat this argument. As the conflicts are cyclic, *Predecessor*(T_q, S_q) used to construct the PGSG when T_q attempts to commit will always contain a non-committed node from T which will then be added as a candidate node to all SSGs at which T_q executed.

Now let T_1 attempt to commit at site S_1 and S_n where $T_1 \rightarrow T_2$ and $T_n \rightarrow T_1$ exist, respectively. Then, as T_n is committed, the SSG at S_n will contain a candidate node – say T_x with respective site S_x – in its *Predecessor*(T_1, S_n). If T_x committed after its propagation to site S_n, then the SSG at S_x would, in turn, contain a candidate node. Finally, as the only node in the cycle that is currently active is T_1, the *Predecessor*(T_1, S_n) constructed at site S_n will contain T_1 as an accessed node as well as a candidate node. Therefore, *Predecessor*(T_1, S_n) will contain the entire cycle. Thus, the PGSG will contain the cycle. As all nodes except T_1 are committed, the cycle will be detected. □

4. Literature review

In this section, a very brief outline of existing techniques that specifically address transaction management in the MMDB environment [3,6,12,14] will be provided and their deficiencies summarized.

The technique proposed in [3] supports two additional types of transactions, namely, reporting transactions and

Table 5
Parameters used for performance analysis.

Parameter	Description	Default values
ST_{avg}	avg. service time of a global transaction	calculated
GT_{exe}	avg. time taken to execute a global transaction	calculated
GT_{commit}	avg. time taken to commit a global transaction	calculated
N_{grp}	avg. number of groups in a global transaction	6 (variable)
EXE_{grp}	avg. time taken to execution a group of site-transactions (includes communication time between the user and MMDBS)	calculated
T_{think}	avg. think time before submitting the next group	0
DCN_{tm}	avg. time between a disconnection and re-connection	1 s
DCN_{dly}	avg. processing delay caused by a disconnection (includes DCN_{tm} and processing time taken to address relocation etc.)	calculated
RL_{tm}	avg. time to address relocation	calculated
DLY_{gpr}	avg. delay added to EXE_{grp} due to disconnection	calculated
DLY_{thk}	avg. delay added to T_{think} due to disconnection	0
P_{dcn}^{grp}	probability of a disconnection occurring during EXE_{grp}	calculated
P_{cn}^{thk}	probability of a disconnection occurring during T_{think}	0
N_{dcn}	avg. number of disconnection for a global transaction	6 (variable)
N_{mgr}	avg. number of migrations for a global transaction	1 (variable)
T_{msg}^{s}	avg. time to transmit a message on the static (wired) network	0.001
T_{sg}^{w}	avg. time to transmit a message over the wireless medium	0.7 s
EXE_{lcl}	avg. local execution time of a site-transaction	0.003 s
P_{cnf}	probability of a site-transaction conflicting with another	0.05 (variable)

co-transactions which allow concurrent global transactions to share partial results improving concurrency. The Multidatabase Transaction Processing Manager technique [14] is based on a Message and Queuing Facility (MQF) that is used to manage global transactions submitted by mobile workstations. The technique presented in [12] is based on an agent-based distributed computing model. Agents may be submitted from various sites including mobile stations. The Kangaroo transaction technique [6] is the only model to capture the movement behavior of the mobile user by viewing mobile transactions from a completely new perspective. A global transaction (a.k.a. Kangaroo transaction) consists of a set of Joey transactions. Each Joey consists of all operations executed within the boundaries of one MSS and is committed independently.

To summarize their deficiencies, none of the reviewed techniques enforce the (global) isolation property. As the isolation property is not enforced, global transactions are not executed as consistent units of computing. In addition, disconnection that represents catastrophic failures is not addressed. It is assumed that disconnection will always be followed by a subsequent reconnection.

5. Performance analysis

In this section, we construct a general MMDB transaction model that is used to study the service time of global transactions in the PS technique and to compare its performance to that of the Kangaroo model. The Kangaroo model is chosen as it is the most recent of the reviewed techniques. Also, this technique supports unrestricted mobility and does not violate local autonomy.

In order to simplify the models, the following assumptions will be made about the environment:

- All sites in the MMDB environment are equally likely to be accessed.
- All messages are of the same size (10 Kb).
- All global transactions are mobile transactions and execute successfully at all sites.
- All site-transactions are equivalent to those specified in TPC-C benchmark.
- Site-transactions of a global transaction are submitted to the MMDBMS in groups. Each group is submitted only after the results of the previous group is received.

The variable used to construct the model and their default values are listed in table 5.

As MMDBS research is a relatively new, values for many of the parameters are not known. For the purpose of this analysis, we have made educated guesses as to their default values. The value of T_{think} has the same effect on all algorithms and, therefore, is set to 0. The value for EXE_{lcl} is obtained from the TPC-C Benchmark [`www.tpc.org`]. We have averaged the response time (obtained from throughput from TPC-C) for five popular databases running on small to medium size servers (IBM DB2 on IBM AS400e, Informix OnLine 7.3 on Compaq ProLiant 5000, MS SQL Server 6.5 on Acer AcerAltos 19000Pro4, Oracle 7.3 on Sun UltraEnterprise 6000, and Sybase SQL Server 11.5 on Compaq ProLiant 6000). Message transmission time has been calculated assuming that the static network is a 10 Mbps Ethernet and the wireless communication medium is cellular telephony with a bandwidth of 14 Kbps [13].

5.1. The general MMDB transaction model

First, we develop a set of general formulas used to calculate the average service time for a mobile global transaction. The service time (ST_{avg}) will be represented using only the major components, i.e., communication time, execution time of site-transactions, the disconnection and relocation time, etc.

ST_{avg} is the time taken to execute a global transaction and the time taken to commit the global transaction. That is,

$$ST_{avg} = GT_{exe} + GT_{commit}. \tag{1}$$

Next, we calculate GT_{exe}. In a static environment

$$GT_{exe} = N_{grp} \cdot EXE_{grp} + (N_{grp} - 1) \cdot T_{think}.$$

That is, the number of groups times the execution time of a group plus the think time between groups. However, in the mobile environment, GT_{exe} is influenced by disconnection. For simplicity, we assume that at most only one disconnection will occur during any GT_{exe}. Then, GT_{exe} is given by

$$GT_{exe} = N_{grp} \cdot \left(EXE_{grp} + P_{dcn}^{grp} \cdot DLY_{grp}\right) + (N_{grp} - 1) \cdot \left(T_{think} + P_{dcn}^{thk} \cdot DLY_{thk}\right). \tag{2}$$

Next, we calculate DCN_{dly}, DLY_{grp}, DLY_{thk}, P_{dcn}^{grp}, and P_{dcn}^{thk}. DCN_{dly} is DCN_{tm} plus the time to address relocation if necessary. That is,

$$DCN_{dly} = DCN_{tm} + \frac{N_{mgr}}{N_{dcn}} \cdot RL_{tm}. \tag{3}$$

DLY_{grp} and DLY_{thk} are influenced by three factors (figure 3):

1 – the total delay caused by disconnection (DCN_{dly});

2 – the point within the current groups execution at which the disconnection occurs (X); and

3 – the length of execution of the current group (EXE_{grp}).

For example, in figure 4(A) DLY_{grp} is 0 and in figure 4(B) DLY_{grp} is $X + DCN_{dly} - EXE_{grp}$.

Note that, a disconnection affects EXE_{grp} only if $X + DCN_{dly} > EXE_{grp}$. Therefore, DLY_{grp} is calculated by taking the probability that $X + DCN_{dly} > EXE_{grp}$ times the average delay to EXE_{grp} given that $X + DCN_{dly} > EXE_{grp}$. Let us consider the cases $DCN_{dly} \leqslant EXE_{grp}$ and $DCN_{dly} > EXE_{grp}$ separately. When $DCN_{dly} \leqslant EXE_{grp}$, the probability that $X + DCN_{dly} > EXE_{grp}$ is DCN_{dly}/EXE_{grp} and the average delay given that $X + DCN_{dly} > EXE_{grp}$ is $DCN_{dly}/2$ – that is the average of the minimum delay (i.e., 0) and the maximum delay (i.e., DCN_{dly}). Thus,

$$DLY_{grp} = \frac{DCN_{dly}}{EXE_{grp}} \cdot \frac{DCN_{dly}}{2}. \tag{4a}$$

When $DCN_{dly} > EXE_{grp}$, the probability that $X + DCN_{dly} > EXE_{grp}$ is 1 and the average delay given that $X + DCN_{dly} > EXE_{grp}$ is $(DCN_{dly} - EXE_{grp} + DCN_{dly})/2$. Thus,

$$DLY_{grp} = 1 \cdot \frac{DCN_{dly} - EXE_{grp} + DCN_{dly}}{2}. \tag{4b}$$

Similarly, to formulate DLY_{thk}, let us consider the case $DCN_{dly} \leqslant T_{think}$ and the case $DCN_{dly} > T_{think}$ separately. When $DCN_{dly} \leqslant T_{think}$ then DLY_{thk} is

$$DLY_{thk} = \frac{DCN_{dly}}{T_{think}} \cdot \frac{DCN_{dly}}{2}. \tag{5a}$$

When $DCN_{dly} > T_{think}$, DLY_{thk} is

$$DLY_{thk} = 1 \cdot \frac{DCN_{dly} - T_{think} + DCN_{dly}}{2}. \tag{5b}$$

The probability of a disconnection occurring during EXE_{grp} (P_{dcn}^{grp}) is

$$P_{dcn}^{grp} = \frac{N_{dcn} \cdot EXE_{grp}}{N_{grp} \cdot EXE_{grp} + (N_{grp} - 1) \cdot T_{think}}. \tag{6a}$$

Similarly, the probability of a disconnection occurring during T_{think} (P_{thk}^{grp}) is

$$P_{thk}^{grp} = \frac{N_{dcn} \cdot T_{think}}{N_{grp} \cdot EXE_{grp} + (N_{grp} - 1) \cdot T_{think}}. \tag{6b}$$

In (4a) and (4b) we have formulated DLY_{grp}, in (5a) and (5b) we have formulated DLY_{thk}, and in (6a) and (6b) we have formulated P_{dcn}^{grp}, and P_{dcn}^{thk}. GT_{exe} can now be obtained from choosing the appropriate formulas for DLY_{grp} and DLY_{thk}. Given GT_{commit} and GT_{exe}, we can calculate ST_{avg} from (1). Note that the values for RL_{tm}, EXE_{grp}, and GT_{commit} need to be calculated for each technique separately.

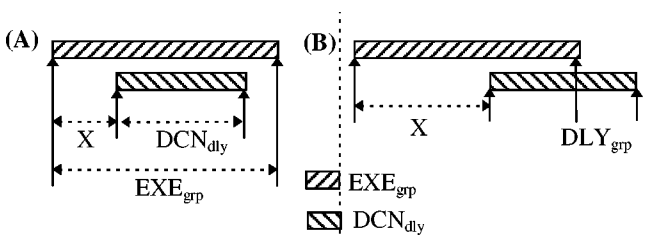


Figure 4. Relationship between DLY_{grp} and X.

5.1.1. The PS technique

In this section, EXE_{grp}, RL_{tm}, and GT_{commit}, for the PS technique will be formulated. First, EXE_{grp} is calculated. For each group of site-transactions, the PS technique will incur two wireless messages to receive a group and submit its outcome to the user. Each site-transaction will require two additional wired messages to submit the site-transactions (and compensating transaction) and receive its outcome. As the site-transactions can be submitted in parallel, EXE_{grp} is given by

$$EXE_{grp} = 2 \cdot T_{msg}^{s} + 2 \cdot T_{msg}^{w} + EXE_{lcl}.$$

In the PS technique, relocation incurs 2 wired messages – one message requesting the global structure and one to

transfer this structure – and one wireless message to re-connect. Therefore,

$$RL_{\text{tm}} = 2 \cdot T^{\text{s}}_{\text{msg}} + T^{\text{w}}_{\text{msg}}.$$

Next, we calculate the GT_{commit}. Let GT_{iso} and GT_{atm} be the average time taken to verify the Isolation property and enforce the Atomicity property respectively. Then,

$$GT_{\text{commit}} = GT_{\text{atm}} + GT_{\text{iso}}.$$

The atomicity property is enforced by sending two message to all sites requesting the status of the site-transactions and sending either an abort or commit. As these messages are sent in parallel,

$$GT_{\text{atm}} = 2 \cdot T^{\text{s}}_{\text{msg}}.$$

Next, we formulate GT_{iso}. To verify serializability of global transaction T_j, the PGSG algorithm requests *Predecessor*(T_j) from all sites at which the global transaction executed its vital site-transactions. These sites will, in turn, request *Predecessor*(T_k) graph for all Candidate nodes T_k in *Predecessor*(T_j). This process continues until there is no candidate node in the Predecessor graph. At each step, for T_k to be a candidate node in *Predecessor*(T_j), three conditions must be satisfied. That is, T_k must conflict with T_j, T_k must have executed prior to T_j at the site at which they conflict, and T_k must be must be active. At each step, as T_k executes prior to T_j, the probability that T_k is active decreases as the time interval since the initiation of T_k increases. Let us assume that, on average, the PGSG algorithm goes through n steps and that at each step the probability that T_k is active decreases evenly, that is $1/n$. Then, as requests for Predecessor graphs are sent in parallel for each candidate node, GT_{iso} is

$$GT_{\text{iso}} = 3 \cdot T^{\text{s}}_{\text{msg}} \cdot P_{\text{cnf}} \cdot \sum_{i=1}^{n} \frac{(n-i)}{n}.$$

As we were unable to obtain values for P_{cnf} and n, we assume that the default $P_{\text{cnf}} = 0.05$ and $n = 4$.

5.1.2. The Kangaroo model

Here we formulate EXE_{grp}, RL_{tm} and GT_{commit} for the Kangaroo Model introduced in [6] executing under the Compensating mode as this mode ensures atomicity. We assume that Joey transactions consist of subtransactions that are analogous to site-transactions.

First, we calculate EXE_{grp}. For each group of site-transactions, the mobile user submits the group to the MSS which, then submits each site-transaction to the respective site (in parallel), receives a response from the site and submits the response to the user. Therefore,

$$EXE_{\text{grp}} = 2 \cdot T^{\text{s}}_{\text{msg}} + 2 \cdot T^{\text{w}}_{\text{msg}} + EXE_{\text{lcl}}.$$

In this model, migration is handled by a hand-off process that requires a HandOff KT (HOKT) record be written to the original MSS's log and a ConTinuing KT (CTKT) record be written to the destination MSS's log. The communication cost of writing the CTKT record is 0. However, to write the HOKT record into the original MSS's log, the doubly linked list that connects the records maintained at each MSS needs to be traversed using one message for each link. For each migration, the average number of links that need to be traversed is $(N_{\text{mgr}} + 1)/2$. In addition, one wireless message is required to re-connect the user. Thus,

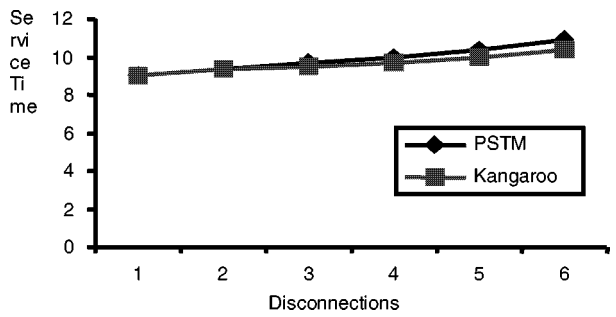
$$RL_{\text{tm}} = T^{\text{w}}_{\text{msg}} + T^{\text{s}}_{\text{msg}} \cdot \frac{N_{\text{mgr}} + 1}{2}.$$

To commit a global transaction, the status table entries for all involved MSSs need to be freed. This requires that the entire doubly linked list related to that global transaction be traversed. Therefore,

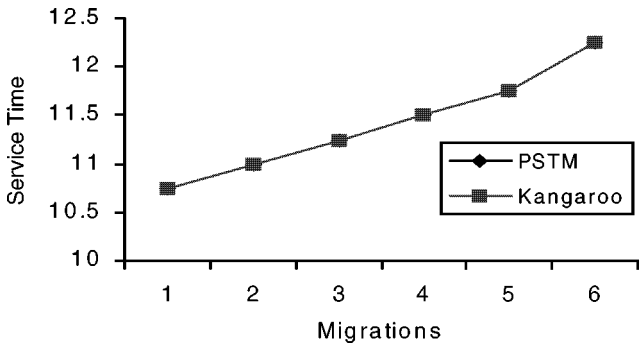
$$GT_{\text{commit}} = T^{\text{s}}_{\text{msg}} \cdot N_{\text{mgr}}.$$

5.2. Comparison results

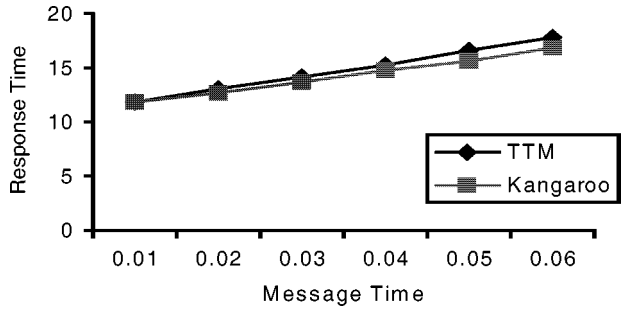
In this subsection, we use the analytical models to examine the performance of the PS technique and to compare it to that of the Kangaroo technique. First, we evaluate each technique with respect to the average number of disconnection and migration for a global transaction. In case 1, we calculate ST_{avg} for different values of N_{dcn} ranging from 1 to 6 using the default values for all other parameters (graph 1). In case 2, we calculate ST_{avg} for different values of N_{mgr} ranging from 1 to 6 (graph 2). Here too, we use the default values for all other parameters except N_{dcn} which is set to 6 as N_{dcn} needs to be greater than or equal to N_{mgr}. In both cases, the evaluation indicates that



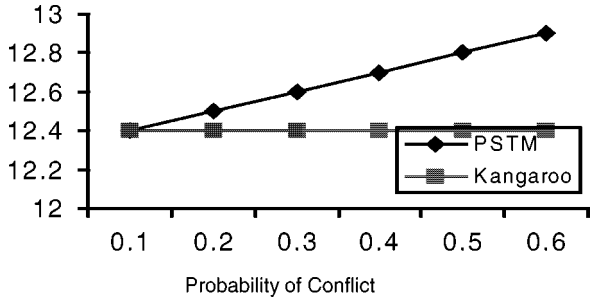
Graph 1.



Graph 2.



Graph 3.



Graph 4.

there is only an insignificant difference between the ST_{avg} of both techniques.

These results suggest that the communication cost incurred by the PS technique in order to enforce the isolation property is minimal. To verify the validity of this conjecture, we need to evaluate the techniques by varying the average communication time on the static network (T^{s}_{msg}) and the probability of conflict for site-transactions (P_{cnf}). If the communication cost incurred by the PS technique to enforce the Isolation property and support LLTs is minimal compared to the Kangaroo technique, then increasing T^{s}_{msg} should have the same effect on both techniques. In case 3, we calculate ST_{avg} for different values of T^{s}_{msg} ranging from the default value of 0.001 to 0.006 (graph 3). This comparison suggests that T^{s}_{msg} has the same effects on both techniques. However, in case 4, when we calculate ST_{avg} for different values of P_{cnf} ranging from 0.1 to 0.6 (graph 4) we see that ST_{avg} of the PS technique is increasingly higher than that of the Kangaroo technique, which stays constant. This suggests two things. First, it suggests that the key component that affects ST_{avg} of the PS technique as opposed to the Kangaroo technique is the probability of conflicts between site-transactions. This is to be expected as the PS technique enforces the Isolation property while the Kangaroo technique does not. Second, it suggests that the PS technique does not perform well in environments with high conflict probabilities between site-transactions. As the enforcement of the Isolation property in the PS technique is based on an optimistic approach, this conforms to the thinking of conventional wisdom where optimistic approaches do not perform well in high conflict environments.

In conclusion, this analysis suggests that the overhead incurred by the PS technique in order to enforce the isolation property and to minimize LLTs being chosen as victims when conflicts do occur, is minimal in MMDB environments that have a low conflict probability between site-transactions.

6. Concluding remarks

In this paper, we propose a Pre-Serialization (PS) technique for the mobile multidatabase systems. This technique allows site-transactions to commit independently so that resources may be released in a timely manner. Two new states – disconnected and suspended – are introduced to fully address disconnection and migration. This technique minimizes the unnecessary abortions caused by erroneous decisions made by the MMDBMS, with respect to the connectivity of the mobile user. A toggle operation is used to minimize the ill-effects of the prolonged execution of long-lived transactions. A PGSG commit algorithm that enforces a wide range of correctness criterion with respect to the atomicity and isolation properties is proposed and its correctness is proved. This algorithm is ideally suited for the MMDB environment as it is de-centralized, and does not require the cooperation of all sites.

We also develop an analytical model to evaluate the performance of the GTM of the MMDBS. We evaluate the performance of the PS technique and compare it to that of the Kangaroo technique. We conclude that the PS technique offers substantial benefits, i.e., fully supports disconnection and verifies Isolation, over existing techniques for very little additional overhead.

Acknowledgement

This material is based in part upon work supported by National Science Foundation under Grant No. EIA-9973465.

References

[1] P. Bernstein et al., The Asilomar report on database research, SIGMOD Record 27(4) (December 1998) 74–80.

[2] Y. Breitbart, H. Garcia-Molina and A. Silberschatz, Overview of multidatabase transaction management, Technical report TR-92-21, University of Texas at Austin (1992) pp. 1–41.

[3] P.K. Chrysanthis, Transaction processing in mobile computing environments, in: *IEEE Workshop on Advances in Parallel and Distance Systems* (October 1993) pp. 77–82.

[4] R. Dirckze and L. Gruenwald, Disconnection and migration in mobile multidatabases, in: *The World Conf. on Design and Process Technology*, Germany (July 1998) pp. 371–377.

[5] W. Du and A.K. Elmagarmid, Quasi-serializability: A correctness criterion for global concurrency control in interbase, in: *Proc. of the 15th Internat. Conf. on VLDB*, Amsterdam, The Netherlands (August 1989) pp. 347–355.

[6] M. Dunham, A. Helal and S. Balakrishnan, A mobile transaction model that captures both the data and movement behavior, Mobile Networks and Applications 2(2) (October 1997) 149–162.

[7] A. Elmargarmid, M. Rusinkiewicz and A. Sheth, *Management of Heterogeneous and Autonomous Database Systems* (Morgan Kaufmann, San Francisco, CA, 1998).
[8] D. Georgakapolous, M. Rusinkiewicz and A. Sheth, On serializability of multidatabase transactions through forced local conflicts, in: *Proc. of the 7th Internat. Conf. on Data Engineering*, Kobe, Japan (1991) pp. 314–323.
[9] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques* (Morgan Kaufmann, San Mateo, CA, 1993).
[10] S. Mehrotra, R. Rastogi, A. Silberschatz and H.F. Korth, A transaction model for multidatabase systems, in: *Proc. of the 12th Internat. Conf. on Dist. Comp. Systems*, Japan (1992) pp. 56–63.
[11] E. Pitoura and B. Bhargava, Dealing with mobility: Issues and research challenges, Technical report CSD-TR-93-070, Purdue University (1993) pp. 1–18.
[12] E. Pitoura and B. Bhargava, A framework for providing consistent and recoverable agent-based access to heterogeneous mobile databases, SIGMOD Record (September 1995) 44–49.
[13] E. Pitoura and G. Samars, *Data Management for Mobile Computing* (Kluwer Academic, Dordrecht, 1998).
[14] L.H. Yeo and A. Zaslavsky, Submission of transactions from mobile workstations in a cooperative MDB processing environment, in: *14th Internat. Conf. on Distributed Computer Systems*, Poland (June 1994) pp. 372–379.

Ravi Dirckze is a Ph.D. candidate in the School of Computer Science at The University of Oklahoma. Ravi received his BS and MS in computer science from the University of Houston, Clear-Lake. He is currently working in the Advanced Technology Group at Unisys Corporation, California. Ravi's major research interest are transaction management and interoperability in heterogeneous databases, mobile databases, and middleware technology.
E-mail: radirckz@cs.ou.edu

Le Gruenwald is a Presidential Professor and an Associate Professor in the School of Computer Science at The University of Oklahoma. She received her Ph.D. in computer science from Southern Methodist University, MS in computer science from the University of Houston, and BS in physics from the University of Saigon. Prior to joining OU, she worked at the Southern Methodist University as a Lecturer in the Computer Science and Engineering Department, and at NEC America, Advanced Switching Laboratory as a Member of the Technical Staff in the Database Management Group. Dr. Gruenwald's major research interests include multimedia databases, distributed databases, mobile databases, object-oriented databases, real-time databases, and data warehouses.
E-mail: gruenwal@cs.ou.edu