Chapter 7

A TAXONOMY OF DATA MANAGEMENTS VIA BROADCASTING IN MOBILE COMPUTING*

Ilyoung Chung, Bharat Bhargava, Sanjay K. Madria

Dept. of Computer Sciences, Purdue University; Dept. of Computer Sciences, Purdue University; Dept. of Computer Science, University of Missouri-Rolla

Abstract: Data management for distributed computing has spawned a variety of research work and commercial products. At the same time, recent technical advances in the development of portable computing devices and the rapidly expanding cordless technologies have made the mobile computing a reality. In conjunction with the existing computing infrastructure, data management for mobile computing gives rise to significant challenges and performance opportunities. Most mobile technologies physically support broadcast to all mobile users inside a cell. In mobile client-server models, a server can take advantage of this characteristics to broadcast information to all mobile clients in its cell. This fact introduces new mechanisms of data management which are different from the traditional algorithms proposed for distributed database systems. In this chapter, we give executive summary and discuss topics such as data dissemination techniques, transaction models and caching strategies that utilize broadcasting medium for data management. There is a wide range of options for the design of model and algorithms for mobile client-server database systems. We present taxonomies that categorize algorithms proposed under each topic. Those taxonomies provide insights into the tradeoffs inherent in each field of data management in mobile computing environments.

Keywords: Broadcasting, Cache Invalidation, Data Dissemination, Transaction management

*This research is supported by CERIAS and NSF grants CCR-9901712 and CCR-0001788

1. INTRODUCTION

The widespread adoption of a client-server system has made this architecture the conventional mode for distributed database systems. At the same time, recent technological advances in the development of portable computing devices and rapidly expanding cordless technologies have made the mobile computing a reality. The combination of these factors gives rise to significant challenges and performance opportunities in the design of mobile client-server database systems. Broadcasting is widely accepted as a medium of disseminating information from the server to mobile clients in a cell. In this chapter, we consider three important topics which utilize broadcasting facility in mobile computing environments: the management of transactions which access data using mobile devices, data dissemination which adopts broadcast as a means of delivering data to mobile devices, and data caching strategies which lessen the dependency on the server.

Approaches to handling weak connectivity in data management systems aim at minimizing communication and surviving short disconnections. However, due to the complicated dependencies among database items, the problem is a complex one. Algorithms on such topics have been proposed to achieve high performance while surviving resource restrictions of resources and connections using broadcasting facility. In this chapter, we present taxonomies that describe the design spaces for transaction management, data dissemination, and caching for data management, and show how proposed algorithms are related to one another. We provide a unified treatment of proposed algorithms that can be used to help understanding the design alternatives and performance tradeoffs for each topic.

The rest of this chapter is organized as follows. In Section 2, alternative architectures for mobile data management are discussed. In Section 3, the taxonomy of concurrency control algorithms which utilize broadcasting is presented and tradeoffs for each level of design options are discussed. In Section 4, we shall focus on mechanisms for data dissemination which broadcasts data to all mobile devices inside a cell. In Section 5, taxonomy for cache invalidation schemes is proposed and discussed, and finally we state our concluding remarks in Section 6.

2. ARCHITECTURAL ALTERNATIVES

A mobile client-server database architecture can be categorized according to some design alternatives, which will be discussed in this section. A mobile client-server database architecture can be categorized according to the four design alternatives namely 1) *the unit of interaction between clients and server*, 2) *data delivery method*, 3) *caching at mobile client*, and 4) *communication*

method. The tradeoffs of these architectural alternatives will be discussed in this section.

2.1 The Unit of Interaction

A mobile client-server database architecture can be categorized according to the unit of interaction between mobile clients and servers. In general, mobile clients can send data requests to the server as *queries* or as *requests for specific data item.* Systems of the former type are referred to as *query-shipping* system and those of the latter type are referred to as *data-shipping*. In query-shipping systems, a client sends a query to the server. The server then processes the query and sends the results back to the client. Queries may be sent as plain text (e.g.,SQL), in a compiled representation, or as calls to precompiled queries that are stored at the server. In query-shipping architecture, communication costs and client buffer space requirements are reduced since only the data items that satisfy a given query are transferred from the server to clients. In contrast, data-shipping system perform the bulk of the work of query processing at the clients, and as a result, much more database functionality is placed at the clients. It offloads functionality from the server to mobile clients. This may prove crucial for performance, as the computing power and the amount of resources at mobile clients get stronger and increase. Also, the frequency of communications between servers and mobile clients reduces, as the applications at mobile clients access data items which resides in the mobile clients. Reduced frequency of communications is crucial for the performance of mobile DBMSs, considering the limited bandwidth of wireless channel. These two alternatives are shown in Fig. 1.

2.2 Data Delivery Method

In a data-shipping client-server system, the server is the repository of the data and the clients are the consumers of the data. Thus, when a client application requires a data item, it has to be delivered from the server to the client. Broadly, there are two ways to achieve the data transfer.

In the client-initiated approach, the client requests a data item from the server on demand, i.e., when the client application requests it. In response to the request, the server locates the data item and transfers it to the client. To make this transfer feasible, the clients and the server typically use a mutually agreed upon request/response protocol. As part of the protocol, the client is allowed to make a predetermined set of request to which the server responds appropriately. On the other hand, in the server-initiated approach, the responsibility of transferring the data rests with the server. Here, the server delivers data items to mobile clients without any explicit request from it, and in anticipation of an access in the future. Thus, unlike in the first approach, the transfer

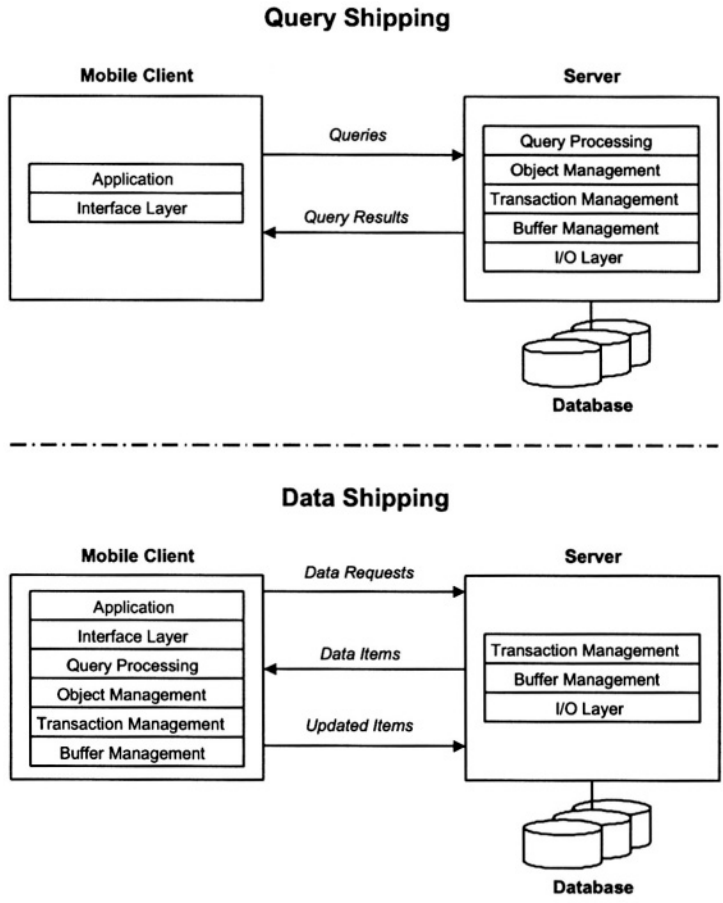

Figure 1. Architectural alternative: Unit of interaction

is initiated by the server and not by the client. Thus the server has to have some knowledge of the client data requirements in this case.

The client-initiated delivery is also called *pull-based delivery* and systems based on it are termed pull-based systems. In effect, in this approach, clients "*pull*" data from the server on demand. Conversely, the data delivery using the server-centric approach is called *push-based delivery*, since the server "*pushes*" data out to clients. These two different alternatives are shown in Fig. 2.

2.3 Caching at Mobile Clients

Client caching refers to the ability of mobile clients to retain copies of data items locally once they have been obtained from the server. In client-server

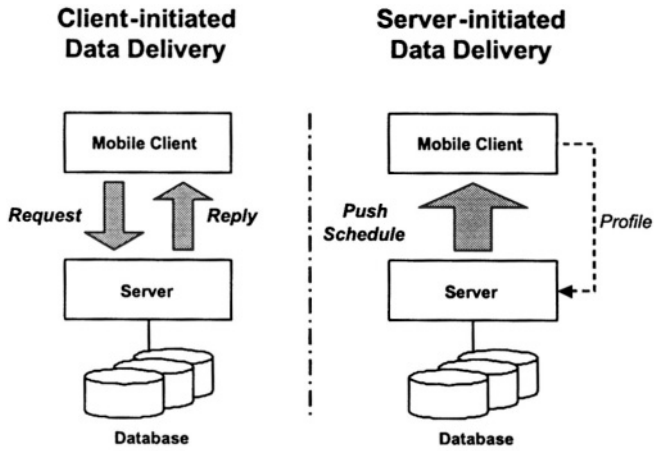

Figure 2. Architectural alternative: Data delivery method

database systems, client caching can be characterized by the following concepts: *dynamic replication* and *second class ownership.* Dynamic replication means that data copies are created and destroyed based on the runtime demands of clients. When a client needs to access a data item, a copy of that item is placed in the mobile client's cache if it does not already exist. In some schemes, data items are removed from a client's cache in order to make room for more recently requested ones because they become invalid. Therefore, the amount of replication that exists in the system at any given time is driven by the recent activity of the mobile clients. This is in contrast to static replication in which the replication of data is determined as part of the physical database design process.

It is well known that replication can reduce data availability in the presence of updates and failures in a distributed environments. Second-class ownership allows consistency to be preserved without sacrificing availability. Second-class ownership refers to the fact that in client caching, the cached copies of pages are not considered to be the equals of actual data items, which are kept at the server.

Mobile client caching is a compromise between the utilization of mobile client resources and the correctness and availability concerns. Mobile client resources are exploited by maintaining the cache, thereby reducing the need to obtain data from the servers. Data that is in a mobile client's local cache can typically be accessed faster than data that is at a server, considering the limited bandwidth of wireless links. Caching, however, does not transfer ownership of data to mobile clients. Servers remain the only true owners of data, and therefore are ultimately responsible for ensuring the correctness of transaction

execution. Client autonomy is sacrificed in the sense that servers maintain data ownership and must remain involved in the execution of transactions. However, the pay off here is that the server's participation in transactions execution is minimized.

2.4 Communication Method

In mobile client-server database systems, the server should maintain communication to mobile clients in order to transfer many control information as well as data items. Those control information are used at mobile clients in processing transactions, which include control of concurrent transactions and maintenance of data consistency. First, the server should send concurrency control information that includes conflict relations of transactions which are executed concurrently at mobile clients. Also, the server should notify a transaction about the commit result of the certification process, according to the correctness criteria. Secondly, update information of data items should be transferred to mobile clients in order to ensure the consistency between the server and mobile clients. When mobile clients maintain local cache and keep a portion of data, applications at mobile clients assume that the local copies of data have up-to-data value of those items. In order to satisfy such assumption, consistency information should be sent to mobile clients, whenever data items are updated.

For sending the control information from the server to mobile clients, there are two communication methods; *unicast* and *broadcast.* With unicast method, the server sends each control information to a specific mobile client, with the knowledge about the clients' information. This method requires that a mobile client register its presence and that a server keep information about mobile clients. The server in this case is *stateful* since it knows about the state of the mobile clients. On the other hand, with broadcast method, the server sends the control information to all the mobile clients periodically or aperiodically, with the same message. Since clients may require different control information simultaneously, the broadcasting message should be well defined. The server in this case is *stateless* since it does not know about the state of mobile clients.

3. TRANSACTION MANAGEMENT

In mobile client-server database systems, transactions are initiated by each mobile client as a string of read and write operations. Because multiple transactions which access common data items may be issued concurrently, there should be a protocol which guarantees the correct execution of concurrent transactions. Several protocols have been proposed in the literature to control concurrent transactions in mobile database systems, and those protocols

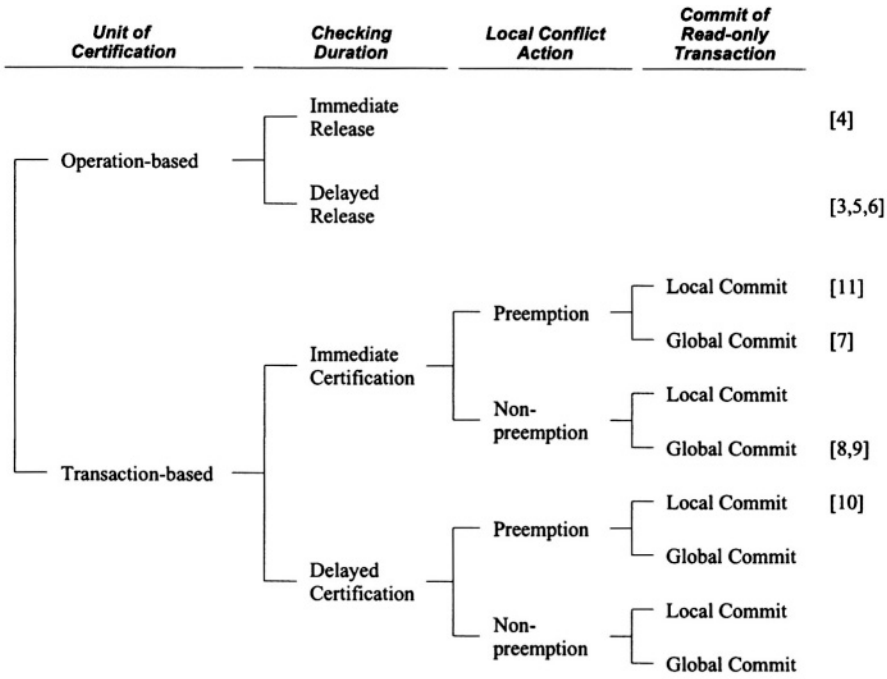

Figure 3. Taxonomy of concurrency control schemes

can be evaluated based on some criteria. In this section, considering the characteristics of mobile environments, we define the following:

- **Concurrency:** The degree of concurrency represents the ability of a protocol which can process multiple transactions simultaneously, without violating the correctness of transactions. Increasing the possible number of concurrent transactions is an important technique to improve the throughput of transaction processing.

- **Autonomy:** Autonomous execution of mobile clients represents the ability to reduce the dependency upon the server while executing transactions. Autonomous execution of transactions at mobile clients can significantly enhance the performance, since it reduces the contention on the bandwidth of wireless link, which is scarce resource in mobile environments.

The design space of concurrency control protocols in mobile database systems is presented in this section, which presents a taxonomy of such protocols.

This taxonomy can be used to help understanding the design alternatives and performance tradeoffs for concurrency control algorithms.

The taxonomy is restricted to algorithms that are applicable to mobile client-server databases and that provide serializability as the correctness criterion. Algorithms that provide lower levels of correctness are not considered here. The taxonomy is shown in Fig. 3. There are a wide range of options for the design of such protocols. We present four levels of classification in the taxonomy, and at the top level of the taxonomy, algorithms are classified according to *the unit of certification* that they employ. This is a fundamental consideration, as it determines when a mobile client initiates the concurrency control action. Three more levels of lower classifications are presented in the taxonomy, but only one of them is applicable to the protocols whose unit of certification is an operation. For the protocols that are classified in the other group at the first level of taxonomy, all of three lower levels, *checking duration, local conflict action* and *read-only transaction* can be applied. Each level of the taxonomy is described in the next subsections.

3.1 Unit of Certification

Mobile clients send the certification request to the server, as transactions initiated at mobile clients should be checked by the server whether they satisfy the correctness criterion. Concurrency control protocols can thus be classified based on the unit of certification sent to the server. The protocols that have been proposed for mobile client-server database systems can be partitioned into two classes based on this criterion: *operation-based* and *transaction-based.*

From another point of view, the difference between the operation-based and transaction-based approaches lies in the type of operations a mobile client expects from the initiated transaction in future [1,2]. If a mobile client thinks that there will be conflicts related to the initiated transaction, it adopts an approach which prevents such conflicts by checking each operation of the transaction, thus it is also called pessimistic approach. On the other hand, if a mobile client expects that most of transactions will not conflict with others; that is there will be few sharing data items in the entire database, it just executes the entire transaction locally. The transaction is checked for consistency preservation only once at the server, thus it is called optimistic approach.

The operation-based scheme requires mobile clients to contact server for every operation of transactions [3,4,5,6]. The server checks if the operation can be processed without violating the correctness criterion, and notifies this information to the mobile client. Then the mobile client can process the next operation. When the mobile client completes all operations of a transaction, the mobile client can decide the commit of the transaction autonomously. This

is because all the operations of the transaction have been verified by the server at the time of request.

In contrast, under the transaction-based scheme, mobile clients are not required to interact with the server while executing transactions [7,8,9,10,11]. Instead, when all operations of the transaction are completed, the mobile client sends the history of the executed transaction to the server in order to verify it. If the server replies that the transaction satisfies the correctness criterion as expected, the mobile client commits the transaction. Otherwise, the transaction should be aborted.

The main argument for the operation-based approach is that it can ensure correct execution. Because each operation of a transaction is checked separately when it is requested, it is impossible for a mobile client to execute any operation which does not satisfy the correctness criterion. Thus, once a transaction completes all the operations, it means that the execution of the transaction ensures the correctness criterion, and it can be committed immediately. As a result, no transaction produces a history which is not serializable. Thus, no transaction is aborted once it completes all operations. (Aborts can occur due to other reasons, such as deadlock between transactions or accessing stale cached data). The disadvantage of the operation-based scheme, however, is a greater dependency on the server. This can result in significant performance overhead, such as increased communication on wireless network.

Avoiding the drawback of the operation-based scheme is the main contribution of the transaction-based scheme. Because a mobile client executes a transaction autonomously until all operations are completed, the transaction-based scheme does not suffer from communication overhead which is fatal in mobile environments. However, this advantage of transaction-based scheme can be achieved at the cost of increased aborts of transactions. If the optimism which is the basis of the transaction-based scheme turns out to be unfounded, then the large portion of transactions which have been executed locally at mobile clients must be aborted. This can degrade the throughput of entire transaction processing.

3.2 Checking Duration

The second level of differentiation for the taxonomy is based on the duration for which the server maintains the checking information of each operation for a transaction (operation-based scheme), or on the point at which the server checks the correctness of transactions (transaction-based scheme).

First, in case of operation-based scheme, conflicting operations of different transactions cannot be granted to be executed concurrently by the server. As a result, the server should prevent two or more operations of conflicting transactions from accessing the same data concurrently. In the operation-based

schemes, whenever an operation is requested by a mobile client, the server gives the right to access the data, which is mutually exclusive (e.g., locking). In the taxonomy, there are two classes of checking duration strategies for the operation-based schemes.

Immediate Release With this strategy, an access privilege of an operation is released immediately after the execution of the operation is completed. As a result, transactions can access a data which has been accessed by a conflicting operation of other transactions which are still active. The main advantage of the immediate release scheme is the higher throughput of the transactions due to the increased concurrency. Since transactions do not hold accessed data until they terminate, more transactions can be executed concurrently. However, in general, accessing data that are written by uncommitted transactions cannot guarantee the correct execution. [4] has proposed *speculative lock management* algorithm as the way of ensuring the correctness of the immediate release scheme. In this algorithm, a transaction releases locks on a data item whenever it writes corresponding data. The waiting transaction reads before- and after- images and carries out speculative execution. In order to satisfy the serializability, the transaction which has carried out speculative executions can commit only after termination of preceding transactions. On the termination of preceding transactions, it selects appropriate execution based on the termination decisions (i.e., commit or abort).

Delayed Release In the operation-based scheme, it is more general to release the access privilege when all the operations of a transaction are completed, and we classified such algorithms as delayed release scheme. With this scheme, once a privilege for a data item is acquired by a transaction, other transactions cannot access that data until the preceding transaction commits or aborts. Although the degree of concurrency is lower compared to the immediate release scheme, it is guaranteed that the schedule produced always ensures the correctness criterion. [5], [6] and [7] proposed concurrency control strategies that use locking with delayed release approach.

On the other hand, several transaction-based protocols have been proposed and studied in the literature. As mobile clients execute the entire transaction locally without any communication with the server, transactions should be checked for correctness before they terminate. Thus, a mobile client sends a messages to the server that request the commit of the transaction. When the server receives such a commit requesting message, it checks if the transaction satifies the correctness criterion. In the taxonomy, we can differentiate the transaction-based scheme into the following two classes according to the point at which the checking is performed.

Immediate Certification With this approach, whenever a mobile client requests commit of a transaction to a server, the server makes the decision of a transaction (i.e., commit or abort) *immediately after* the server receives commit requesting message [7,8,9]. The main argument for the immediate certification scheme is simplicity. Because certification actions of a transaction T_i in this case is performed against the transactions which have requested commit earlier than T_i, the server does not have to consider transactions whose commit requesting messages arrive after T_i. As a result, in this scheme, the order of arrival of commit requesting message is the main parameter which decides the commit or abort of the transaction.

Delayed Certification In the transaction-based scheme, the server can delay the decision of commit requesting transactions until the result of decision is actually sentsentsent to mobile clients. The delayed certification scheme was proposed and studied in [10], in order to increase concurrency of transactions. With the delayed certification process, the server can select transactions which are related to a large number of conflicts, and by aborting such transactions, the throughput can be improved. The delayed certification scheme can be applied when the server sends the results of certifications to mobile clients periodically.

3.3 Local Conflict Action

The next level of differentiation for the taxonomy is local conflict action, which is applicable only to the transaction-based scheme. This level is based on the priority given to the committing transactions at remote mobile clients to which they are sent. When the committing transaction shows a conflict with an active transaction at a mobile client, there are two options: preemption and non-preemption.

Preemption With the preemption scheme, ongoing transactions at mobile clients are aborted as the result of an incoming transaction which shows conflicts. Under this scheme, the optimism that is assumed in mobile clients regarding the execution of a transaction is somewhat weaker than under the non-preemption scheme. This is because the non-preemption scheme will force a committing transaction to serialize behind a locally ongoing transaction if a conflict is detected, whereas under the preemption scheme, committing transactions always have priority over ongoing transactions, so conflicting local transactions are aborted. When an active transaction which conflicts with committing transactions completes all operations, it can not be committed by the server, as it performed operations that conflict with already committed transactions. Thus, aborting such local transactions early in mobile clients can reduce unnecessary communication overhead. The preemption scheme was firstly proposed in [7], which proposed a concurrency control protocol called *Wound Certifier*. The

key to the algorithm is the use of the broadcast channel to transmit information about read and write sets so mobile clients can decide whether their active transactions can continue or should be aborted. By checking in every period whether the current read and write set of the transaction intersects with those of committed transactions, the mobile client is acting as a *Wound Certifier* for its own transactions. The decision of aborting a transaction is done if the transaction's read or write set intersects with a committing transaction's read or write sets. [11] also has proposed two preemption schemes, versioning and invalidation method, which can abort active transaction at mobile clients using information broadcasted from the server. In this way, the algorithm downloads some of the work of validating transactions to the mobile clients, as a result, can increase the autonomy.

Non-preemption In contrast to the preemption scheme, with the non-preemption scheme, committing transactions does not preempt the ongoing conflicting transaction. Ongoing transactions continue their operations regardless of the committing transaction which conflicts. Then, most of such active transactions cannot be committed when they complete their operations. Although the procedure at mobile clients is simple with non-preemption scheme, it is quite wasteful to send a commit request message of a transaction that cannot be committed.

3.4 Commit of Read-only Transactions

When all operations of a transaction are read operations, we can consider a special commit process which is performed independently by mobile clients. We classify the concurrency control protocols according to the commit process of read-only transactions. This level of differentiation also is only applicable to the transaction-based scheme. In case of the operation-based scheme, as each operation should be guaranteed by the server, it is impossible to apply local commit policy.

Local Commit of Read-only Transactions As described in the previous section, using the transaction-based scheme, transactions executed locally at mobile clients must be sent to the server to be checked for the correctness. If mobile clients can decide commit or abort of a locally executed transaction only with information which is broadcasted from the server (without uplink message), the overall pathlength of transactions can be significantly shortened, and the throughput can be improved through the offloading of the wireless network. Although transactions which updated data items should be sent to the server because of the update installation, special consideration can be given to read-only transactions [10,11]. With the local commit policy for read-only transactions, a mobile client commits a transaction autonomously, if all opera-

tions of the transaction are read operations. Of course, there should be a special consideration to commit a read-only transaction locally, in order to ensure the correctness criterion.

Global Commit of Read-only Transactions If a protocol has no consideration for the local commit of read-only transactions, all transactions should be sent to the server to be guaranteed the correctness. Most of transaction-based protocols proposed in the literature adopted the global commit strategy for read-only transactions.

4. DATA DISSEMINATION

Traditionally, the mode of data delivery has largely been on request-response style. Users explicitly requests data items from the server. When a data request is received at a server, the server locates the information of interest and returns it to the client. This form of data delivery is called pull-based. In wireless computing environments, the stationary server machines are provided with a relatively high bandwidth channel which supports broadcast delivery to mobile devices in their cell. As a result, in recent years, different models of data delivery have been explored, particularly the periodic push-based model where servers repetitively broadcasts data to mobile clients without any explicit requests.

In mobile computing environments, several criteria can be used to evaluate the performance of a data delivery method.

- **Responsiveness:** The most important criterion of a data delivery scheme is its ability to get the requested data to the user quickly. In this regard, two metrics can be considered. The first one is the average access time, which is the amount of time spent, on average, from the instant the request is made to the time that the requested data item is received. The second metric is the worst-case access time that measures the maximum access time for any user request to be satisfied.
- **Scalability** In mobile client-server systems, one of the most important criteria is the cell capacity that measures the number of mobile devices which can be handled by a server. The effectiveness of a scheme is also determined by how well it adapts to workload or environmental changes. The scheme should be able to support increasingly large number of population of users.
- **Power Efficiency** As battery power is a precious resource for mobile devices, it has to be minimized.
- **Tuning Time** Another metric that is commonly used as an indication of the energy consumption of an data delivery strategy is the tuning time.

The tuning time measures the amount time that a mobile client listens to the channel. Thus, it measures the time during which the mobile client stays in the active mode and therefore determines the power consumed by the client to retrieve the relevant data.

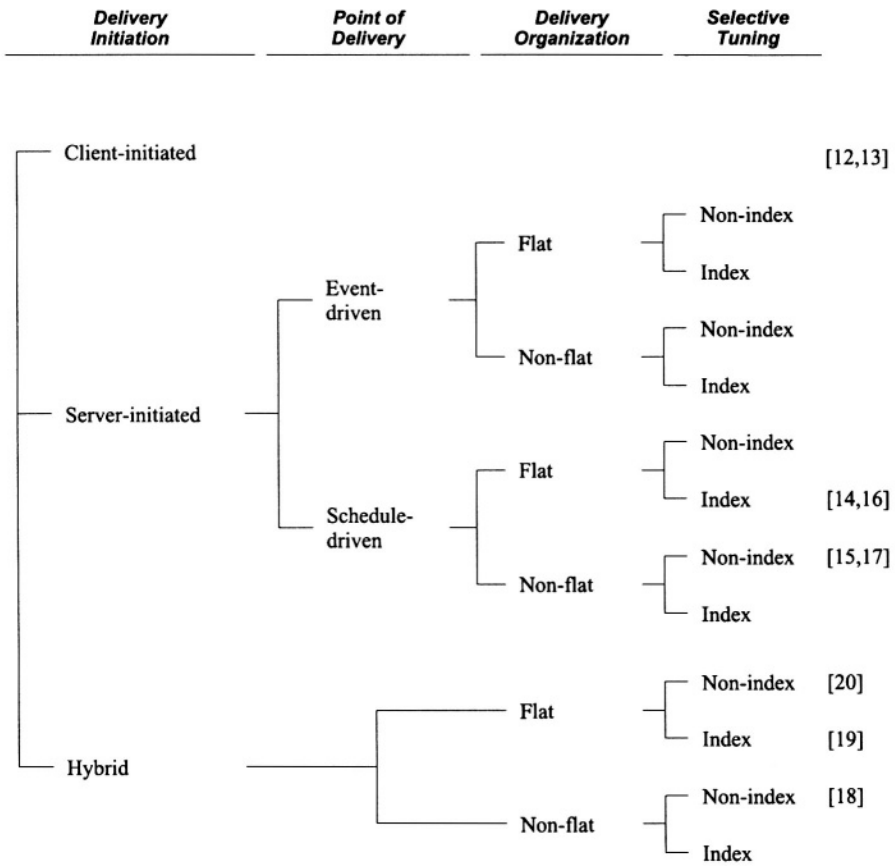

Figure 4. Taxonomy of data delivery schemes

We shall present an alternative taxonomy of data delivery mechanisms based on the design issues that need to be addressed in developing those schemes. This taxonomy is restricted to algorithms in the literature that were proposed for the mobile computing environments. The taxonomy is shown in Fig. 4. We present also four levels of the classification that can be broadly categorized as delivery initiation, point of delivery, delivery organization and selective tuning. The top level of classification is fundamental consideration, as it determines

which part is obligated to initiate the delivery of data items. Three more levels of lower classifications are presented in the taxonomy, and all of them are applicable to the protocols whose delivery is initiated by the server. Each level of the taxonomy is described in the following.

4.1 Delivery Initiation

Data delivery can be initiated by the mobile client (client-initiated) or by the server (server-initiated). Under the client-initiated approach, mobile clients pull the desired data object from the server by submitting queries to the server, the server accesses the relevant data items from the database and returns them to the mobile clients. The server initiated approach operates in a different manner. The server initiates data delivery and sends data objects to mobile clients, possibly without any explicit requests from clients. In this case, to receive the data items, mobile clients must listen to the broadcast channel to filter the incoming stream for their desired data.

Client-initiated Delivery In traditional client-server systems, data are delivered on a demand basis. A client explicitly requests data items from the server. When a data request is received at a server, the server locates the data of interest and returns it to the mobile client. This form of data delivery is called client-initiated or pull based data delivery. The client-initiated approach is effective when the client population is small; otherwise, the communication channel and the server can become a bottleneck quickly. Moreover, it requires an uplink channel to be available and that the mobile client must have transmission capacity. In addition, mobile clients need to be able to know what they want to retrieve. The client-initiated data delivery is adopted in [12,13].

Server-initiated Delivery In the server-initiated approach, the responsibility of transferring data rests with the server. With this approach, the server delivers data items to mobile clients without any explicit request from it, and in anticipation of an access in the future. Thus, unlike in the pull based approach, the transfer is initiated by the server and not by the client, and this approach is called server-initiated approach or push based data delivery. The server has to have some knowledge of the client data requirements for this method to work well. Server-initiated approach is an effective mechanism for large client population, and avoids the limitations of the client-initiated schemes in mobile computing environments. However, it is also limited with a problem that it is difficult to predict accurately the needs of mobile clients. Sending irrelevant data results in poor use of the channel bandwidth, and data may not reach the mobile clients in time. The server-initiated data delivery has been explored in [14,15,16,17].

Hybrid Delivery Push and pull based delivery can be combined by considering systems in which besides the broadcast channel, mobile clients are provided with an uplink channel from the clients to the server, also called back channel, used to send messages to the server. This approach is called hybrid data delivery. In a hybrid approach, some data items are delivered by the server initiation, while the remaining data items are to be requested by mobile clients before they are delivered. One important factor in hybrid delivery is whether the same channel from the server to the clients is used for both broadcast delivery and for the transmission of the replies to on demand requests. In this case, techniques for efficiently sharing the channel are of interest. Mobile clients can use back channel to provide feedback and profile information to the server. They can also use the back channel to directly request data, for instance critical data for which they cannot wait to appear on the broadcast [18].

Instead of broadcasting all data items in the database, one way to realize hybrid data delivery is to partition data items into two sets: one being broadcasted and the other being available only on demand [19]. Determining which part of the database to broadcast is a complicated task since the decision depends on many factors including mobile clients' access patterns and the server's capacity to service requests.

Mobility of users is also critical in determining the set of broadcast items. Cells may differ in their type of communication infrastructure and thus in their capacity to service requests. Furthermore, as users move between cells, the distribution of requests for specific data at each cell changes. An adaptive algorithm that takes into account mobility of users between cells is proposed in [20].

4.2 Point of Delivery

This dimension of data delivery examines the schedulability of the data, i.e., whether the data items are delivered based on event-driven or schedule driven. This level of differentiation is only applicable to the server-initiated delivery scheme or the hybrid delivery scheme. In case of the client-initiates scheme, as the server delivers data items in response to the requests from mobile clients, it is impossible to apply schedule-driven policy.

Event-driven In data delivery scheme which adopts event-driven strategy, there is no predetermined schedule on how data items are to be delivered. Data items are disseminated in response to events such as requests or triggered by updates. Thus all the proposed client-initiated schemes are classified as event-driven data delivery.

Schedule-driven Data delivery schemes, which adopt schedule-driven policy, deliver data based on some predetermined schedule. For example, information

may be sent out daily or weekly, or information may be polled periodically as in the remote-sensing application.

4.3 Delivery Organization

The data items to be delivered have to be organized for dissemination. The data items to be delivered may be organized in consideration of bandwidth utilization and performance improvements. Mobile clients are interested in accessing specific data items from the delivered data. The access time is the average time elapsed from the moment a mobile client expresses its interest to an item by submitting a query to the receipt of the item on the broadcast channel. The broadcasted data should be organized so that the access time is minimized. This level of classification is based on the strategy to organize the contents of information which is delivered by the server, and as a result, this lever is applicable to schedule-driven data delivery schemes. In the schedule-driven scheme, data items disseminated follows a regular, repeating program, and this program may be flat or non-flat.

Flat The simplest way to organize the transmission of broadcast data is a flat organization. In flat programs, all data items are of equal importance, and broadcasted once in a broadcast cycle [14,16]. Given an indication of the data items desired by each mobile client listening to the broadcast, the server simply takes the union of the required items and broadcast the resulting set cyclicly. The regularity of a flat organization makes it easier to design mechanisms that allow mobile clients to search the desired portion of data.

Non-flat The basic idea of the non-flat organization is to broadcast data items that are most likely to be of interest to a larger part of the client community more frequently than others. Thus, the non-flat program favors objects with higher access frequencies. Hence, in a broadcast cycle of a non-flat organization, while all data items are broadcasted, some will appear more often than others. Doing so, in effect, creates an arbitrary fine grained memory hierarchy as the expected delay in obtaining an item depends on how often that item is broadcasted. Non-flat programs yields shorter access time for popular data items as compared to flat programs at the expense of longer access time for data items that are less frequently accessed. Non-flat programs also provide a better average access time than flat programs. However, it has higher bandwidth requirement as its broadcast cycle length is longer than than of flat programs. Non-flat strategies have been explored in [15,17].

4.4 Selective Tuning

In a server-initiated delivery mechanism, a mobile client listening to the channel needs to examine every data item that is broadcasted. The tuning time of data delivery is the amount of time spent listening to the broadcast channel. Listening to the broadcast channel requires the mobile client to be in the active mode and increase power consumption. Some mechanisms that adopted indexing have been proposed to minimize this power consuming process, which is scarce resource at mobile device, thus the last level of differentiation is selective tuning.

Non-index With the non-indexing data delivery mechanism, the server just broadcast data items in flat or non-flat organization. Mobile clients should then listen to the broadcast's channel until they obtain requested data items. This process requires the CPU to be in the active mode, which is a power consuming operation at mobile device. Since the mobile client is typically interested in only a small subset of the broadcasted data, the overhead of scanning the other objects is wasted.

Index Mobile clients may be interested in fetching from the broadcast individual data items identified by some key. To minimize the scarce energy resources in mobile devices, methods to index data have been proposed so that mobile clients only need to selectively tune to the desired data [14,16,19]. Thus, most of the time clients will remain in doze mode and thus save energy. The objective is to develop methods for allocating indeces together with data on the broadcast channel so that both access and tuning time are optimized.

5. CACHE CONSISTENCY

The bandwidth of the wireless channel is rather limited, and as a result, *caching* of frequently accessed data in a mobile client can be an effective approach for reducing contention on the narrow bandwidth wireless channel. Caching allows the database systems to use the resource of mobile clients in order to reduce the number of data requests that must be sent to the server. The effectiveness of caching depends on the assumption that there is significant *locality of access* in the system workload. Locality can be considered along two dimensions:

- *Temporal Locality*: References to items are clustered in time. If an item is accessed, it is likely to be accessed again in the near future.
- *Spatial Locality*: References to items are clustered in space. If an item is accessed, it is likely that items that are physically near it will be referenced in the near future.

However, once caching is used, a *consistency maintenance* strategy is required to ensure the consistency of cached data. Because data items are allowed to be cached by multiple mobile clients, a mechanism for ensuring that all mobile clients see a consistent view of database should be used. This is referred to as the *cache consistency* problem. This is, unfortunately, difficult to enforce in a mobile computing environments due to the frequent disconnection and mobility of clients. Clients who resume connection no longer know whether their cached content is still valid.

Traditional techniques require either the server to transmit invalidation messages to the clients every time an object is updated or the mobile client to query the server to verify the validity of the cache contents. Both approaches, however, are not adequate for mobile computing environments. In the first approach, which has also been referred to as stateful based approach, the server must keep track of the mobile clients' cache content and locate the appropriate clients whenever a data item is updated. Moreover, even if a mobile client is not using a particular cached data item, it gets notified about its invalid status, which is a potential waste of bandwidth. In the second approach, the mobile client must send a message every time they want to use their cache. This is both wasteful of bandwidth and battery power of mobile devices.

This section provides a taxonomy of consistency maintenance protocols that encompasses the algorithms proposed in the literature. The taxonomy is restricted to algorithms that are applicable to mobile client-server database systems. In stateless-based cache consistency schemes, the server has no information about which clients are currently under its cell and what data items are cached by mobile clients. Most of cache consistency schemes proposed in the literature are stateless-based. The taxonomy is shown in Fig. 5. We present three levels of classification in the taxonomy, and at the top level, algorithms are classified according to by whom *the consistency action* is *initiated.* This is a fundamental consideration, as it determines which part is obligated to maintain the consistency. Each level of the taxonomy is described in the subsections that follow.

5.1 Consistency Action Initiation

Because cached data is replicated data, it follows that traditional methods for managing updates to replicated data can be used or extended to manage cached copies at mobile clients. Cache consistency maintenance protocols can thus be classified based on the result of a particular update. The protocols that have been proposed for mobile client-server databases can be partitioned into two classes based on this criterion: *client-initiated* and *server-initiated.*

From a qualitative point of view, the difference between the client-initiated and server-initiated approaches lies in how access to stale data is prevented.

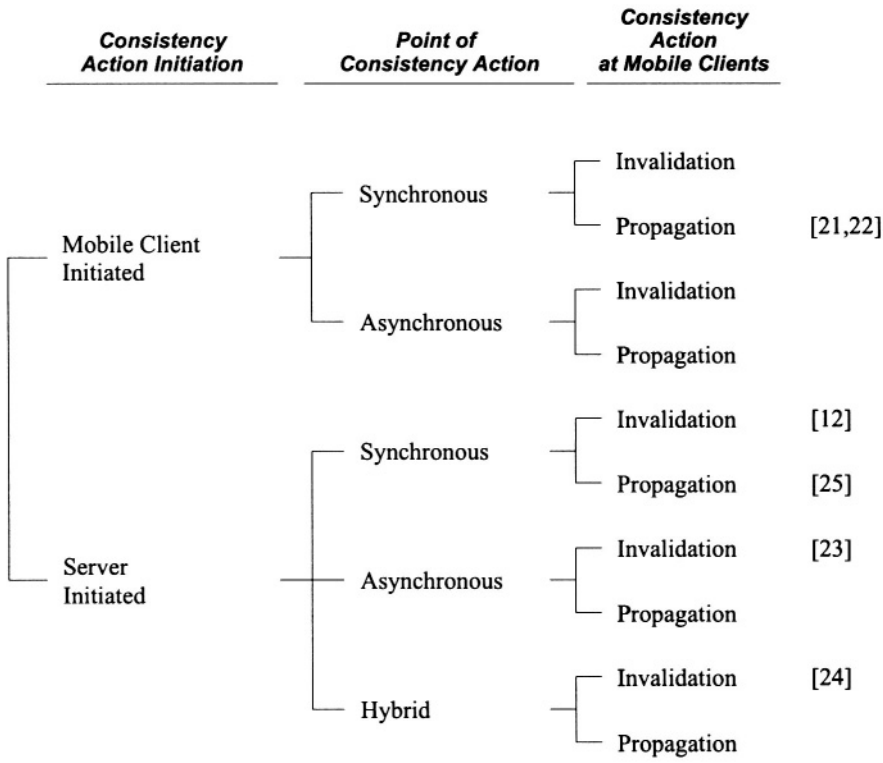

Figure 5. Taxonomy of cache consistency schemes

Specifically, a data item is considered to be stale if its value is older than the item's latest committed value. Consider the concurrency control algorithms proposed in the previous section, in which data items are tagged with sequence numbers, and where the sequence number of an item is increased when a transaction that has updated the item commits. A copy of a data item is then considered to be stale, if its sequence number is lower than the latest sequence number assigned to any copy of the data item.

The *client-initiated* scheme allows stale data copies to reside in a mobile client's cache. The validity of cached data items are checked when they are actually used by a transaction [21,22]. As a result, mobile clients are responsible for checking the consistency of data. The client-initiated scheme is so named because the validity checking of cached data is explicitly performed by mobile clients.

In contrast, under the *server-initiated* scheme, the consistency of data copies at mobile clients is maintained by the server, with periodic or aperiodic broad-

casting messages [12,23,24,25]. As a result, most of cached data at mobile clients are likely to have up-to-date value. However, the consistency of cached data is not strictly guaranteed, because there exists an interval for a committed updated transaction until it is notified to mobile clients.

The main argument for the client-initiated scheme is that mobile clients cannot execute transactions with stale data copies, as mobile clients check the validity of data whenever they are accessed by transactions. With the client-initiated scheme, stale data copies in mobile clients' cache are detected immediately during the execution of transactions, thus aborts can be avoided. The disadvantage of the client-initiated scheme, however, is increased communications between mobile clients and servers, especially uplink communications which occurs whenever cached copies are accessed. This can result in significant performance overhead, considering the asymmetric environment of wireless channel.

On the other hand, the server-initiated scheme does not require any uplink communication for the consistency check, although the consistency of cached data is not strictly guaranteed. Because the server is responsible for maintaining the consistency, mobile clients just listen for consistency information broadcasted from the server, instead of sending messages to check the validity. If a transaction accesses data which has been updated in the server. If the mobile client is not yet informed the update, the transaction will be aborted, as the inconsistency is detected at the end of the transaction.

5.2 Point of Consistency Action

The second level of differentiation for the taxonomy is based on the strategy which is adopted to check the validity of data items which are accessed by an active transaction (client-initiated scheme), or to send information about updated data items by the server (server-initiated scheme).

At first, the client-initiated scheme can be classified based on the time when mobile clients check the consistency of data touched by a transaction. The client-initiated scheme does not guarantee the consistency of data in mobile clients' cache, and as a result, the consistency of data items accessed by a transaction must be determined before the transaction can be allowed to commit. Thus, the consistency checks should begin and complete during the execution of a transaction. In the taxonomy, there are two classes of consistency checking strategies: *Synchronous* and *Asynchronous.*

Synchronous Checking On the first access that a transaction makes to a particular data item, the mobile client must communicate with the server to ensure that its copy of the item is valid [21,22]. In immediate checking scheme, this is done in a synchronous manner - the transaction is not allowed to access the data item until its validity has been verified. Once the validity of the mobile

client's copy of the data has been established, the copy is guaranteed to remain valid at least until the transaction completes. The main argument of the immediate checking scheme is the strict consistency of accessed data. When a transaction completes all operations with the immediate checking scheme, it is always guaranteed that all data accessed by the transaction is valid, and as a result, can be committed. However, such guarantee of consistency is achieved at the cost of increased communications with the server.

Asynchronous Checking Delayed checking is an optimistic approach compared to the immediate checking. No consistency action is sent to the server until the transaction has completed its operations and is ready to commit. At this point, information on all the data items read and written by the transaction is sent to the server, and the server determines whether or not the transaction should be allowed to commit. This scheme can have two advantages. First, consistency actions can be bundled together in order to reduce consistency maintenance overhead. Secondly, any consistency maintenance work performed for a transaction that ultimately aborts is wasted; delayed checking can avoid some of this work. The main disadvantage of the scheme is, however, that the delay can result in the late detection of data conflicts. The resolution of inconsistency that are detected after they have occurred typically requires aborting one or more transactions.

On the other hand, in the server-initiated scheme, the server sends the consistency information to mobile clients by broadcasting messages, and in the taxonomy, we classified the server-initiated scheme into three classes based on the point at which such broadcasting messages are sent: *synchronous broadcasting, asynchronous broadcasting* and *hybrid broadcasting.*

Synchronous Broadcasting The synchronous broadcasting method is based on periodic broadcasting of invalidation reports. The server keeps track of the data items that are recently updated, and broadcast these information periodically. With this scheme, at a periodic broadcasting point, the server sends the list of data items which have been updated after the last broadcasting point. Since the broadcasting occurs periodically, the message overhead for the synchronous broadcasting scheme can be stable, and as a result, the communication overhead on wireless network is relatively low. However, it cannot adapt to the update frequency, and as a result, many cached copies at mobile clients can have inconsistent value, when frequent updates by transactions exist. A mobile client has to listen to the report to decide whether its cache is valid or not. Thus, each mobile client is confident for the validity of of its cache only as of the last invalidation report. That adds some latency to query processing, since to answer a query, a mobile client has to wait for the next invalidation re-

port. This overhead in query latency can be avoided if a less strict consistency model is adopted. Three synchronous broadcasting strategies have been proposed in [12]. In the broadcasting timestamps strategy, the invalidation reports contain the timestamps of the latest change for data items updated in the last w seconds. In the amnestic terminals strategy, the server only broadcasts the identifiers of data items changed since the last invalidation report. In the signature strategy, signatures are broadcasted. A signature is a checksum computed over the value of a number of data items by applying compression technique.

Asynchronous Broadcasting The consistency information can be broadcasted immediately after changes to data items occur, in order to adapt to update frequency by transactions. This approach is called the asynchronous broadcasting scheme, which is proposed in the following section. With the asynchronous broadcasting scheme, the server can adjust the broadcasting period according the frequency of updates on data items, and as a result, is effective for connected mobile clients, and allows them to be notified immediately of updates. However, for a mobile client who reconnects after a period of disconnection, the client has no idea of what has been updated and so the entirety of its cache content has to be invalidated. An asynchronous technique based on bit sequences has been proposed in [23]. In this strategy, the invalidation report is organized as a set of bit sequences with an associated set of timestamps. Each bit in the sequence represents a data item in the database. The set of bit sequences is organized in a hierarchical structure. It is shown that the algorithm performs consistently well under conditions of variable update rate and client disconnection time.

Hybrid Broadcasting The hybrid broadcasting scheme is an hybrid approach between synchronous and asynchronous broadcasting. In order to adapt to the caching pattern of each data item, this scheme selects asynchronous broadcasting for widely cached data items. and synchronous broadcasting for exclusively used data items [24]. As a result, this scheme can reduce communication overhead which is indispensable for the asynchronous broadcasting, while still adapting to the update frequency.

5.3 Consistency Action at Mobile Clients

The next level of differentiation for the taxonomy is consistency action which is performed at mobile clients when they receive the broadcasted consistency information. This level of differentiation is applicable only to the server-initiated schemes, and there are two options here: *invalidation* and *propagation.*

Invalidation With the invalidation scheme, when mobile clients receive the list of updated data items which is broadcasted from the server, they remove the stale copy of data items from the cache, so it will not be accessed by any subsequent transactions. After a data item is invalidated at a mobile client, a subsequent transaction that wishes to access the data item at that mobile client must obtain a new copy from the server. The invalidation scheme is adopted in most of cache consistency algorithms proposed for mobile environments [12,22,23,24].

Propagation Propagation results in the newly updated value being installed at the mobile client in the place of stale copy [21,22,25]. In this way, mobile clients do not have to request newly updated data to the server. However, transmitting the updated value of data items may be an overhead in terms of wireless communication. Thus, most of propagation schemes are proposed for the traditional wired client-server database systems. There is a tradeoff between invalidation and propagation. Under the propagation strategy, when disconnection time is short, mobile clients can update their cache immediately. Under invalidation strategy, mobile clients must still submit requests to retrieve the updated records even if the disconnection is short. However, under the propagation strategy, since the entire content of a data item is broadcasted, the report is much larger and can take up a significant portion of downlink channel capacity which is a scarce resource in mobile computing environments.

6. CONCLUSIONS

This chapter investigated a range of data management techniques which utilize broadcasting facility in mobile computing environments. Three main research topics were addressed: concurrency control, data dissemination and cache consistency. Broadcasting approach to transmit information to numerous concurrent mobile clients is attractive in mobile computing environment, because a server need not know the location and the connection status of its clients, and because the clients need not establish an uplink connection which is expensive in asymmetric communication environment. In this chapter, we presented design spaces of various algorithms proposed in those research topics. These taxonomies can be used to help understanding the design alternatives and performance tradeoffs of those algorithms.

REFERENCES

[1] R.E. Gruber, "*Optimism vs. Locking: A Study of Concurrency Control for Client-Server Object-Oriented Databases*," Ph.d. Thesis, Dept. of

Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1997.

[2] I. Chung, J. Lee and C.-S. Hwang, "A Contention Based Dynamic Consistency Maintenance for Client Cache," in *Proc, of International Conference on Information and Knowledge Management,* 1997, pp. 363-370.

[3] J. Jing, O. Bukhres and A. Elmgamid, "Distributed Lock Management for Mobile Transactions," in *Proc. of IEEE International Conference on Distributed Computing Systems,* 1995, pp. 118-125.

[4] P. Reddy and M. Kitsuregawa, "Speculative Lock Management to Increase Concurrency in Mobile Environments," in *Proc. of International Conference on Mobile Data Access, Lecture Note in Computer Science,* vol. 1748, Springer, 1999, pp. 82-96.

[5] Q. Lu and M. Satyanarayanan, "Resouese Conservation in a Mobile Transaction System," *IEEE Transactions on Computer,* vol. 46, no. 3, 1997, pp. 299-311.

[6] A.K. Elmagarmid, J. Jing and O.A. Bukhres, "An Efficient and Reliable Reservation Algorithm for Mobile Transactions," in *Proceedings of International Conference on Information and Knowledge Management,* 1995, pp. 90-95.

[7] D. Barbara, "Certification Reports: Supporting Transactions in Wireless Systems," in *Proceedings of IEEE International Conference on Distributed Computing Systems,* 1997, pp. 466-473.

[8] V.C.S. Lee and K.-W. Lam, "Optimistic Concurrency Control in Broadcast Environments: Looking Forward at the Server and Backward at the Clients," in *Proceedings of International Conference on Mobile Data Access, Lecture Note in Computer Science,* vol. 1748, Springer, 1999, pp. 97-106.

[9] J. Shanmugasundaram, A. Nithrakashyap and R. Sivasankaran, "Efficient Concurrency Control for Broadcast Environments," in *Proceedings of ACM SIGMOD International Conference on Management of Data,* 1999, pp. 85-96.

[10] I. Chung and C.-S. Hwang, "Increasing Concurrency of Transactions using Delayed Certification," in *Proceedings of International Conference on Mobile Data Management,* 2001, pp. 277-278.

[11] E. Pitoura and P.K. Chrysanthis, "Exploiting Versions for Handling Updates in Broadcast Disks," in *Proceedings of International Conference on Very Large Databases*, 1999, pp. 114-125.

[12] D. Barbara and T. Imielinsky, "Sleepers and Workaholics: Caching Strategy in Mobile Environments," *VLDB Journal,* vol.4, no.4, 1995, pp. 567-602.

[13] K.L. Tan and B.C. Ooi, "Batch Scheduling for Demand-driven Servers in Wireless Environments," *Information Sciences,* vol. 109, 1998, pp.281-198.

[14] T. Imielinski, S. Viswanathan and B.R. Badrinath, "Energy Efficient Indexing on Air," in *Proceedings of ACM SIGMOD International Conference on Management of Data,* 1994, pp. 25-36.

[15] N. Vaidya and S. Hameed, "Scheduling Data Broadcast in Asymmetric communication environments," *ACM/Baltzer Wireless Networks,* vol. 5, no. 3, 1999, pp. 171-182.

[16] E. Pitoura and P.K. Chrysanthis, "Scalable Processing of Read-Only Transactions in Broadcast Push," in *Proceedings of International Conference on Distributed Computing Systems,* 1999, pp. 432-439.

[17] S. Acharya, R. Alonso, M.J. Franklin and S.B. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communications Environments," in *Proceedings of ACM SIGMOD International Conference on Management of Data,* 1995, pp. 199-210.

[18] S. Acharya, M. Franklin and S. Zdonik, "Balancing Push and Pull for Data Broadcast," in *Proceedings of ACM SIGMOD International Conference on Management of Data,* 1997, pp. 183-194.

[19] K. Stathatos, N. Roussopoulos and J.S. Baras, "Adaptive Data Broadcast in Hybrid Networks," in *Proceedings of International Conference on Very Large Data Bases,* 1997, pp. 326-335.

[20] A. Datta, A. Celik, J. Kim, D. Vander and V. Kumar, "Adaptive Broadcast Protocols to Support Efficient and Energy Conserving Retrieval from Databases in Mobile Computing Environments," in *Proceedings of International Conference on Data Engineering,* pp.124-133, 1997.

[21] M.H. Wong and W.M. Leung, "A Caching Policy to Support Read-only Transactions in a Mobile Computing Environment," Technical Report, Dept. of Computer Science, The Chinese Univ. of Hong Kong, 1995.

[22] W.-C. Peng and M.-S. Chen, "A Dynamic and Adaptive Cache Retrieval Scheme for Mobile Computing," in *Proceedings of IFCIS International Conference on Cooperative Information Systems,* 1998, pp. 251-259.

[23] J. Jing, A. Elmagarmid, A. Helal and A. Alonso, "Bit Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments," *Mobile Networks and Applications,* vol. 2, no. 2, 1997, pp. 115-127.

[24] I. Chung, J. Ryu and C.-S. Hwang, "Efficient Cache Management Protocol Based on Data Locality in Mobile DBMSs," in *Current Issues in Databases and Information Systems, Proceedings of Conference on Advances in Databases and Information Systems, Lecture Note in Computer Science,* vol. 1884, Springer, 2000, pp. 51-64.

[25] J. Cai, K.L. Tan and B.C. Ooi, "On Incremental Cache Coherency Schemes in Mobile Computing Environment," in *Proceedings of International Conference on Data Engineering,* 1997, pp. 114-123.