APPLYING GENERALIZED SNAPSHOT ISOLATION TO MOBILE DATABASES

ABSTRACT

Mobile devices that host embedded database systems are becoming a commonality. Transaction management is needed to provide mobile data availability and transaction processing functionality during disconnection periods. In this paper, we present an adaptation of the transaction correctness criterion GSI to mobile networks that reduces the number of message exchanged per transaction. We discuss the advantages of our approach for mobile networks and applications.

KEYWORDS

Generalized Snapshot Isolation; Multi-version Concurrency Control; Mobile Transaction Management; Mobile Databases; Database Correctness Criterion.

1. INTRODUCTION

We are witnessing a torrential development of mobile devices, networks, software and applications, many of which rely on stored data. However, factors such as scarce bandwidth, handover problems, lack of standard compatibility with changing network carriers and backbones, limited client resources etc remain barriers for a full utilization of mobile computing capabilities. For mobile databases, there is yet another potential impediment: they need to be able to deal with temporary disconnections from the network. Disconnection means that communication with other network nodes is not possible. That is either due to a deliberate decision of the device, network operator or user to interrupt or leave an ongoing network session, or it is caused by an asynchronous breakdown of nodes or connections between them [Phatak & Badrinath, 1999]. In any case, it is desirable that mobile users can continue their work also when they are disconnected. In particular, updates performed by disconnected users are logged and later propagated to servers. Several problems may arise: (1) updates by different disconnected users may conflict among each other; (2) due to that, it is often impossible to predict the result of such updates; (3) mobile users may wish to perform operations on remote data that are not locally replicated. There are several proposals to manage mobile database transactions, trying to provide a serializable isolation level [Bernstein et al, 1987]. Just to mention a few: the use of speculative locking as in [Reddy & Kitsuregawa, 2004], multiversioning [Bernstein et al, 1987] and locking protocols [Madria et al, 2002] [Phatak & Badrinath, 1999]. In general, the strong correctness criterion of one-copy serializability (1CS) [Bernstein et al, 1987] needs to be compromised for mobile settings.

Caching of frequently accessed data plays an important role in mobile computing since it alleviates limitations of performance and availability during weakly connected and disconnected time intervals. Approaches to provide a consistent serial cache in mobile nodes [Lee et al, 1999] or buffering at the base station level [Peng & Chen, 2005] have been made, with broadcast messages instead of caches. Since an entire large database typically cannot be cached in a mobile device, transactions have to be very simple, e.g. as realized by stored procedures [Preguiça et al, 2000].

An example of a mobile application is a warehouse salesmen force, equipped with mobile devices that retrieve a list of clients in the morning and update client purchases offline; connections are established at the end of the day, so as to minimize conflicts. Other examples can be imagined for e-commerce, or guided tourist tours where conflicts are rare and possibly concurrent updates are not severe. Hence, strong serializability does not seem to be best suited for such kind of applications, even more so when Two Phase Locking (2PL) [Bernstein et al, 1987] is the concurrency control protocol of choice, whereas most commercial database vendors provide snapshot isolation (SI) [Berenson et al, 1996]. 2PL provokes a lot of transaction blocking, which is especially harmful where disconnections are frequent, so that SI seems to be preferable. With SI, users see the latest snapshot of committed data, hence a transaction will not block during read operations. Generalized SI (GSI) extends SI to distributed settings [Elnikety et al, 2005], however without guaranteeing the latest version: transactions read from an older but consistent snapshot, while maintaining the properties of SI. So, the number of messages can be significantly reduced; only transactions with write access (update transactions) need to involve the base station.

In this paper, we propose a transaction management algorithm providing GSI. The mobile device will cache a portion of the database. This assumption is reasonable, given that there are benchmark tools such as TPC-W [TPC, 2006] that allow for tailoring the workload and possible user interactions, so that caching makes sense. Our algorithm reduces the interaction with the database server, since only update transactions will involve the database server. Furthermore, each base station will be augmented with a DBMS that contains a (possibly older) consistent copy of the database. The rest of the paper is organized as follows. Section 2 explains the system model used. Section 3 recapitulates different correctness criteria for centralized and distributed databases. The motivation and description of our algorithm is outlined in section 4. Previous related work is addressed in section 5. Section 6 concludes the paper.

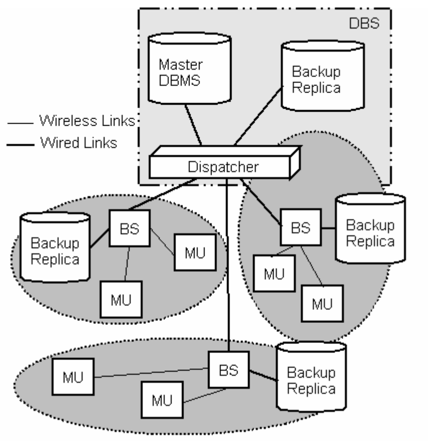

Figure 1. An abstraction of the mobile system architecture

2. SYSTEM MODEL

Figure 1 depicts the system architecture. We assume there is one master Database Server (DBS, hereafter) with one Database Management Systems (DBMS), and several more DBMSs acting as backup copies for fault tolerance. This DBMS provides SI as the transaction isolation level. The DBS interacts with Mobile Hosts (MH) issuing Mobile Transactions (MT). The MH communicates with the DBS by way of a Base Station (BS). During the MT transaction lifetime, the MH may move and switch from one BS to another (*hand-over*) or even disconnect from the network (due to economical factors, unavailable connectivity or the application model). The communication between a MH and its BS is performed by an unreliable wireless link. On the other hand, the communication between the BS and the DBS is through reliable channels. In the following, we are going to develop each component in detail.

2.1 Database Server (DBS)

We briefly address the following components: dispatcher, master database and (one or several) backup replica(s). The dispatcher receives requests from various BSs. These requests exclusively consist of the set of tuples (i.e., key/value pairs) modified by the MT (i.e., the *writeset* of the MT). They are directly forwarded to the master database that will then issue the transaction. The dispatcher receives the outcome of the transaction and forwards it to the BS, where the MH is located. Moreover, if the final decision is to commit the transaction, the latter will be multicast to the rest of BSs. Replication of the master data in all replicas can be achieved as outlined in the Ganymed approach [Plattner et al, 2006].

2.2 Base Station (BS)

The BS serves and manages the MHs that are currently located in its cell. It receives transactions from the MHs and sends them to the DBS. They also hold a copy (possibly outdated) of the database, so as to reduce overhead of the centralized DBS for serving database snapshot versions. This also exemplifies the general benefit obtained by having a database copy located near to the final users, so that they have access to a version of the database without interacting with the DBS.

2.3 Mobile Hosts (MH)

The MH has strict requisites of power consumption and cache, due to hand-off and the network bandwidth. We assume that a MH has a mobile processor with some database functionalities. The MH is the responsible for creating MT in behalf of the final users. At this point it is of key importance to highlight the benefits of exploiting the application dependence, such as locality aware, e-commerce transaction, etc. Furthermore, it is important to consider whether these MTs are writing intensive or not, their consistency needs and the degree of conflicts with other transactions among other factors. Of course, the MH does not hold a whole snapshot copy of the database; it only stores part of the database accessed by the user (a cache of part of the database version gotten from the BS). There are several examples of these, such as finding a restaurant at a given cell or the type of transition established in TPC-W [TPC 2006] for e-commerce in an electronic bookstore.

An MT consists of a set of data access operations (read and/or write), followed by a commit or abort operation. We assume that all of them run under GSI. A MH maintains two copies of the data: committed and tentative. The committed version of data contains (possibly out of date) data directly received from the BS. The tentative version is based on the committed data and it reflects the execution of previously submitted MT in the MH. While disconnected, application may access both versions but users must be aware of this fact or mark the start of the transaction [Preguiça et al, 2000]. The way the MT is executed will be discussed in more detail in Section 4.

3. CORRECTNESS CRITERIA IN DISTRIBUTED DATABASES

There are several correctness criteria in the literature for replicated databases. Most of them are conceptually reduced to the one copy equivalence principle of a centralized scheduler, i.e., several physical copies of the same object viewed as a single logical object. Therefore, we start with an outline of the centralized scenario (serializable and SI) and then move on to the distributed case.

3.1 Centralized Databases

A *transaction* groups a set of read and write operations (i.e., insert, update and delete statements) that transforms the database from one consistent state to another. A *history* models the interleaved execution of a set of transactions. Two actions in a history are said to *conflict* if they are performed by distinct transactions on the same data item and at least one of is a write operation. The most straightforward solution is to implement serial scheduling by way of locks, more precisely 2PL.

However, weaker consistency levels have been considered for certain applications (e.g., dynamic web content generation), for obtaining a higher throughput, since a lot of transactions will remain blocked. Weaker consistency naturally suggests a weaker isolation level, such as Snapshot Isolation (SI). In SI, each transaction T reads data from a snapshot of the (committed) data as of the time the transaction started (T_{start}), which may be any time before the transaction's first read. A transaction running in SI is never blocked while attempting a read statement; since they read from their snapshot and see their own updates (insertions, deletions and modifications) in their snapshots. Updates by other transactions active after T_{start} are invisible to T. The SI is a kind of multiversion concurrency control. The commitment of a transaction abides by the *first-committer-wins* rule, getting a commit timestamp T_{commit}, which is greater than any existing one. The transaction successfully commits only if there is no other transaction T' with a commit timestamp in the interval [T_{start} , T_{commit}] that wrote data that T also wrote. Otherwise, T will abort.

3.2 Distributed Databases

For the distributed case, we focus on One Copy Serializable (1CS), Generalized Snapshot Isolation (GSI), and One Copy Snapshot Isolation (1CSI) schedules [Lin et al, 2005] [Gonzalez de Mendívil et al, 2006]. All of them are meant to be used with DBMSs at each node providing different transaction isolation levels such as serializable (1CS) and SI (resp., GSI and 1CSI).

3.2.1 One Copy Serializable (1CS)

1CS is the strongest correctness criterion for replicated databases. Replication is transparent to the execution of a transaction. Its interleaved execution with other transactions in the system is equivalent to a serially ordered execution of the transactions in a centralized database, i.e., the effect of transactions is the same at all available replicas.

3.2.2 Generalized Snapshot Isolation (GSI)

In [Elnikety et al, 2005], the main principles of SI are postulated. Essentially, each read operation sees data from a snapshot of data committed by the time it has started. The cited paper also reports on GSI as a first attempt to port SI to a replicated setting. Simultaneously, [Lin et al, 2005] generalized SI to the replicated case for middleware architectures, calling it One Copy Snapshot Isolation. Thus, in GSI, a transaction can observe an older database snapshot (i.e., the local replica snapshot). Writes performed by the transaction are more likely to be outdated, especially as the transaction takes a very old snapshot, and will get aborted at commit time by a *newer* transaction. GSI is defined due to the fact that writesets cannot be immediately applied in all replicas at a time and, due to this, the snapshot being used in a transaction might be older than the one that would have been assigned to it according to physical time in a hypothetical centralized system. Yet, all relevant properties of SI still hold for GSI, in particular those that guarantee a serializable behavior.

For a read-only transaction, commitment will be immediate, even though it has read from an older snapshot. For a transaction with non-empty writeset, a certification process is needed in a database running GSI, i.e., the database certifies whether *T* can commit or needs to abort. Thus, it dynamically enforces that no other transaction has updated data items that *T* has updated too. More precisely, the database checks whether *writeset(T)* intersects with the writeset of any transaction that committed after T_{start}. If all intersections are empty, *T* will commit; otherwise *T* will abort.

3.2.3 One Copy Snapshot Isolation (1CSI)

1CSI names a family of variants of GSI where transactions in a distributed database read the latest version. More precisely, we refer to 1CSI as the attempt to port SI properties to a distributed setting, i.e., for working with the latest snapshot. One approach to obtain SI in a replicated setting is by sending the readsets and writesets of transactions to the DBS. Additionally, the replication protocol must wait for the latest version of the database; otherwise, it will not do better than GSI, given that transactions are atomically committed in total order at all nodes, as observed in [González de Mendívil et al, 2006]. However, this approach is not feasible in a mobile network, since sending the read set to all sites is costly, leading to lower performance, poor scalability and higher abortion rates. And even for fixed networks, this option is discarded in [Gray et al, 1996] when that paper discusses 1CS, since 1CS requires the same support as 1CSI (i.e., both readsets and writesets must be transferred).

4. TRANSACTION MANAGEMENT ALGORITHM

In this section, we motivate the need of a specific transaction management for mobile networks in general, and describe a suitable replication protocol in particular.

4.1 Motivation

No matter which application, transactions in a mobile network are characterized by some common features. They are firstly executed at the MH and are later validated by the DBS. It is reasonable to assume that MTs prefer to use committed data, although at first, they need to connect to the BS to retrieve the currently stored version. We assume that transactions conform to certain patterns of data accesses and transitions, as assumed in TPC-W [TPC, 2006]. Hence, if most of the operations are read-only, then the interaction with the other replica will be kept to a minimum, so that the most suitable correctness criterion is GSI. (The amount of messages and the information that would be needed to maintain 1CSI disqualifies it as a valid approach for mobile networks.)

Consider a mobile user visiting a city, interested in, e.g., hotel locations, special events, certain restaurants or cinema ticketing information. Another question is whenever a MT wants to modify the database state; it is needed to interact with the DBS to issue a certification process of the MT executed. On the other hand, working with tentative data is best suited for disconnected environments. However, this may lead to cascading aborts [Bernstein et al, 1987] since new MT may read data from an invalid snapshot. NH working with tentative data may receive information of the outcome of an update transaction by additional means present in mobile communications (e.g. short messaging and paging services). Therefore, our main goal is to ensure a given correctness criterion (GSI) that minimizes the number and size of message exchanges. Since, only a few times a copy of the portion of the database accessed must be downloaded, and the messages upload is reduced to a few tuples of the updated items (writeset) otherwise none. The main advantage of this approach is that with GSI we still achieve the main properties of SI and with the appropriate serializability conditions, either static or dynamic ones; it is possible to generate serializable histories while keeping the number of messages to a minimum.

4.2 Transaction Execution

Let us begin with the assumption that the MH connects for the first time to the system. An MH starts the execution of a MT in the system by contacting the BS of its associated cell. This initial message will contain details about the user identifier (billing and authorization issues) and the kind of query (e.g., the top 25 bestsellers of an online bookshop). The BS will answer by sending the requested content, thus permitting the execution of the transaction along with the snapshot version of the database from where they were taken. Of course, the BS will maintain a transaction in its underlying DBMS associated to this MT, since additional information may need to be retrieved from the same snapshot version; however, for simplicity, we assume in this paper that only one message is needed, if at all, with the BS, and we do not go into further details regarding this issue.

The user may read or write data in the context of that MT. Read-only accesses can be committed without further ado, while for the commitment for writes, the writeset of the MT, along with the snapshot version, is sent to the BS. The BS acts as a repeater and sends the message to the DBS that executes the GSI certification process in the master database for transaction MT [Elnikety et al, 2005]. Recall that we are passing the writeset as key/value pairs, and that the snapshot version of the MT may be very old, so that the same data item(s) may have been updated already before. In such a case, the transaction is rolled back and an abort message is sent to the BS. Otherwise, the transaction is committed, and the database sends a commit message to the BS. The BS forwards the commit or abort message to the MH. In case of commit, its cached (tentative) version then becomes the committed state. If an abort message is received, then all transaction executed at the MH after the commit request of MT must be rolled back too (i.e. cascading aborts for transactions issued at the MH with tentative data are not avoided).

Note that MH roaming does not affect the transaction management. Forwarding the commit or abort message is affected only if there is a hand-over during the certification process of the MT. Moreover, as pointed out before, the validation of the transaction may be notified via a SMS message to the final user of the transaction if the MH remains disconnected [Preguiça et al, 2000].

Once a MT has been committed at the DBS, the scheduler multicasts the writeset of the certified transaction along with its T_{commt} timestamp to the rest of BS, using the FIFO communication service [Chockler et al, 2001]. In turn, the BSs update their database copy in an ordered fashion so that they keep a consistent (older) snapshot version of the database. This last point constitutes the key idea about how GSI is guaranteed with this transaction execution schema. As updates are certified in a single node, the commit operations are ordered, and since they are propagated according to that order, GSI is guaranteed.

Finally, several optimizations may be considered, even though they may at first not appear as realistic or valid. One is that the DBS can be one of the BS itself. Its state may be maintained by a failure detector for the rest of BSs [Chandra & Toueg, 1996], so that one of the remaining BSs can replace the crashed BS as the new DBS. Another improvement is to implement a full replicated certifier for the entire system. However, that does not scale well, due to the need of applying all transaction updates at all available BSs. Moreover, both solutions only may make sense for WLANs, but not for public mobile networks since, in densely populated areas with communication micro-cells, replication and failure management at this level is not feasible [Gray et al, 1996].

5. RELATED WORK

In this section we will briefly overview some previously proposed transaction management solutions for mobile environments. Traditionally, mobile transaction management has been conceived as an extension of distributed transaction management in fixed network. However, divergent characteristics for both kind of networks have been pointed out in [Gray et al, 1996], such as the limited bandwidth and frequent disconnections in the mobile case that may necessitate special reconciliation techniques. Subsequently we are going to sketch two mechanisms for achieving serial executions by using locking schemes. In [Reddy & Kitsuregawa, 2004], a speculative locking mechanism consisting in modifying the write lock to a short lock of the 2PL is described. This means that the write lock is released releases the lock on the data object whenever it produces corresponding after-image during its execution; the mechanism works under the assumption that almost every transaction wishes to commit no-matter-what, so read-only transactions may read from both versions, which leads to a complex mechanism of cascading aborts. Moreover, read-only transactions have to wait for the commitment of previous transactions and even for the commitment of transient values. [Madria et al, 2002] propose a modification of 2PL by adding a *validate* lock, which contains a committed but not validated version. In both mechanisms, timestamps are used to prevent deadlocks in the locking mechanism. The way transactions are executed is as a stored procedure where MTs must wait for acquiring all the proper locks before issuing the operations at the MH. Our approach is totally different; it exchanges fewer messages with the DBS and provides a completely different approach avoiding all the blocking time wasted at the DBS, thanks to the GSI level. Multiversion reconciliation fro providing serializability is introduced in [Phatak & Badrinath, 1999] in which serializabilty is ensured at the server side by computing and finding the proper snapshot accommodation, although in case of conflict the reconciliation is left to the application.

Driven by burgeoning market needs, also commercial vendors have begun to offer solution for mobile databases. Oracle Lite 10g [Oracle, 2006] presents a similar schema to the one presented in this paper. The main difference consists in the way snapshots are propagated: Oracle needs to provide different snapshots for different purposes (e.g., read-only, updatable, etch), whereas we provide a generic partial snapshot of the database to the MH. The MT is free to update it or not. Moreover, conflict detection is independent of the application nor does it rely on the system administrator, since it is governed by the certification process [Elnikety et al, 2005]. Currently, a growing trend toward middleware support [Beigl, 1997] and service-oriented architectures [Gollmick, 2004] for mobile databases can be observed.

6. CONCLUSIONS

We have adapted an existing correctness criterion of distributed databases (GSI) to be usable in mobile networks. GSI is a relaxed form of SI where transactions may see an “older” snapshot while maintaining all properties of SI. This approach is especially useful for mobile environments since the number of exchanged messages per transaction is significantly reduced. Most of the limitations of bandwidth and memory associated with mobile devices may be overcome with this solution. Furthermore, dynamic web content generation and several business applications are expected to work very well with this kind of isolation level.

ACKNOWLEDGEMENT

This work has been supported by the Spanish Government under research grant TIN2006-14738-C02.

REFERENCES

Beigl, M., 1997. MODBC - A Middleware for Accessing Databases from Mobile Computers. *Proceedings of the 3rd Cabernet Plenary Workshop*. Rennes, France.

Berenson, H. et al., 1995. A Critique of ANSI SQL Isolation Levels. *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*. San Jose, California, USA, pp. 1-10.

Bernstein, P.A. et al., 1987. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, Reading, Massachusetts, USA.

Chandra, T.D., and Toueg S., 1996. Unreliable Failure Detectors for Reliable Distributed Systems. *In ACM Journal*, Vol. 43, No. 2, pp. 225-267.

Chockler, G., et al., 2001. Group communication specifications: a comprehensive study. *In ACM Comput. Surv.*, Vol. 33, No. 4, pp. 427-469.

Elnikety, S. et al., 2005. Database replication providing generalized snapshot isolation. *Proceedings of the IEEE Symposium on Reliable Distributed Systems*. Orlando, Florida, USA, pp. 73-84.

Gollmick, C., 2004. Client-Oriented Replication in Mobile Database Environments. *Technical Report MINET 03-08, Friedrich-Schiller-Universität Jena*, Jena, Germany. `http://www.minet.uni-jena.de/Math-Net/reports/sources/2003/03-08report.ps`

González de Mendívil, J.R. et al., 2006. Non-Blocking ROWA Protocols Implement Generalized Snapshot Isolation Using Snapshot Isolation Replicas. *Technical Report ITI-ITE-06/04, Instituto Tecnológico de Informática*, Valencia, Spain. `http://www.gsd.unavarra.es/gsd/proyectos/spora/files/TR-ITI-ITE-0604.pdf`

Gray, J. et al., 1996. The Dangers of Replication and a Solution. *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*. Montreal, Quebec, Canada, pp. 173-182.

Lee, S.K., et al., 1999. Supporting Transactional Cache Consistency in Mobile Database Systems. *Proceedings of the of the ACM International Workshop on Data Engineering for Wireless and Mobile Access*. Seattle, Washington, USA, pp. 6-13.

Madria, S.K., et al., 2002. A Multi-version Transaction Model to Improve Data Availability in Mobile Computing. *Proceedings of the CoopIS/DOA/ODBASE Conference, volume 2519 of LNCS*. Irvine, California, USA, pp. 322-338.

Oracle, 2006. Oracle Database Lite. `http://www.oracle.com/technology/products/lite/index.html`

Patiño-Martínez, M. et al, 2005. MIDDLE-R: Consistent database replication at the middleware level. In ACM Trans. Comput. Syst., Vol. 23, No. 4,pp 375-423.

Peng, W.C., Chen, M.S., 2005. Design and Performance Studies of an Adaptive Cache Retrieval Scheme in a Mobile Computing Environment. *In IEEE Trans. Knowl. Data Eng.*, Vol. 4, No. 1, pp. 29-40.

Phatak, S.H., and Badrinath B.R., 1999. Multiversion Reconciliation for Mobile Databases. *Proceedings of the International Conference on Data Engineering.* Sydney, Australia, pp. 582-589.

Plattner, C., et al., 2006. Extending DBMSs with Satellite Databases. *In VLDB Journal. Accepted for publication.*

Preguiça, N.M. et al., 2000. Mobile Transaction Management in Mobisnap. *Proceedings of the ADBIS-DASFAA Conference, volume 1884 of LNCS*. Prague, Czech Republic, pp. 379-386.

Reddy, P.K., and Kitsuregawa, M., 2004. Speculative Locking Protocols to Improve Performance for Distributed Database System. *In IEEE Trans. Knowl. Data Eng.*, Vol. 16, No. 2, pp. 154-169.

TPC, 2006. Transaction Processing Performance Council. `http://www.tpc.org`