Isolating Transactions on Replicated Content Going Mobile*

José Enrique Armendáriz-Iñigo, Hendrik Decker, Francesc D. Muñoz-Escoí
Instituto Tecnológico de Informática, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{armendariz, hendrik, fmunyoz}@iti.upv.es

Abstract

Mobile databases are a centerpiece of the dramatic growth of data-centric applications for mobile computing, nomadic communication and wireless networks. Replication of database content is useful in client-server and peer-to-peer configurations of mobile distributed databases. We present an adaptation of generalized snapshot isolation for transactions on content data in mobile networks. This solution overcomes various limitations of known approaches to manage mobile database transactions.

1. Introduction

Hardware and bandwidth capacities of devices and networks are continuing to grow at phenomenal rates. Thus, increasingly vast amounts of storage as well as more and more applications (both conventional and innovative) can be embedded on platforms that hitherto had been considered too limited for accommodating advanced services. As a result, mobile communication devices such as circuit-switched or VoIP-based mobile phones, internet-enabled PDAs, handheld computers and wireless laptops are increasingly converging in capacity and functionality with more conventional computing systems. One such functionality is content data processing.

A cornerstone of many data-centric applications is a mobility-enabled database that stores content data. We say "content data" for distinguishing it from signalling, protocol control and other meta data that may need to be kept persistently in some storage device. Our main focus, however, is on evolving content data and the associated problems of keeping it available and up-to-date. For convenience, we are going to simply speak of "data", from now on, whereby we always mean content data.

Data evolve via transactions. Thus, in mobile networks, a transaction management is needed that is capable of providing data availability and transaction processing functionality also during disconnection periods. For networks and devices with sufficient storage and processing capacity, replication has been an approved means for achieving availability and reliability. Given the ever-growing capacities of small devices as observed above, it should therefore be a natural consequence to use replication technology also for mobile content. A major hurdle to do so, however, is that transaction properties such as ACID, or weaker ones as associated to snapshot isolation (SI), can no longer be guaranteed for mobile networks, where disconnection periods can be arbitrary long.

In this paper, we present an adaptation of Generalized Snapshot Isolation (GSI) to mobile replicated database transactions. It overcomes various limitations of other correctness criteria, by a possibly significant reduction of the number of message rounds per transaction. We discuss its advantages for mobile networks and content data-centric applications. In Section 2, we outline the assumed basic system architecture. In Sections 3 and 4, we discuss the necessity of relaxing some correctness criteria for the management of mobile transactions over replicated data. In Section 5, we describe an algorithm for adapting GSI to mobile replication platforms. In Section 6, we address related work. In Section 7, we conclude.

2. System Architecture

As shown in Figure 1, we assume a single database server (DBS), consisting of a DBMS and a local backup, plus backup copies at several distributed base stations (BS). The DBMS provides SI as the transaction isolation level. The DBS interacts with Mobile Hosts (MH), from which Mobile Transactions (MT) are initiated. Each MH communicates with DBS via a BS. During the lifetime of a MT, MH may be on the move, being handed over from one BS to another, or being disconnect, maybe on purpose or due to some failure. Communication between MH and BS is wireless and unreliable, while communication channels between BS and DBS are reliable.

*Supported by FEDER and the Spanish MEC under research grant TIN2006-14738-C02.

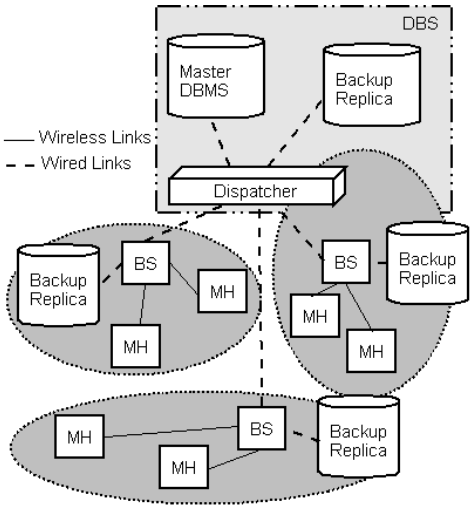

Figure 1. System Architecture

2.1. Database Server (DBS)

The dispatcher receives requests from various BSs, each consisting of the updates (i.e., writeset) of a MT, i.e., key/value pairs as modified by MT. They are directly forwarded to the master DBMS that will then process the transaction. The dispatcher receives the outcome of the transaction and forwards it to the BS that currently covers the MH. Moreover, if the transaction is to be committed, it will be multicast to all other BSs. Replication of the master data in all replicas can be achieved as outlined in the Ganymed approach [16].

2.2. Base Station (BS)

The BS serves the MHs that are currently located in its cell. It receives transactions from the MHs and forwards them to DBS. The MHs hold a (possibly outdated) copy of the database, for reducing the overhead at the DBS for providing database snapshots. Moreover, to have a local database copy near to the end users enables them to access data without having to interact with the remote DBS.

2.3. Mobile Hosts (MH)

MHs may suffer strong constraints with regard to power consumption, cache limitations, hand-overs and available bandwidth. Each MH can be assumed to have a local mobility-enabled processor with some database functionality. MH takes care of creating MTs on behalf of the end users. Apart from specific application needs, current location parameters and other mobility-relevant issues, a smart transaction management will also take into account whether transactions are read- or write-intensive, as well as what their consistency requirements and conflict ratios with other transactions are. Of course, the MH typically does not hold a complete copy of the database, but only cashed parts obtained from the BS that are accessed by its end users. Execution of MTs will be discussed in Section 5.

3. Mobile Transactions

Mobile transactions over replicated content data need to be able to deal with temporary disconnections from the network, when communication with other nodes is not possible. That is either due to the user's or application's deliberate decision, or it is caused by an asynchronous interrupt or crash [15]. Mobile users and applications then will want to continue their work. Updates by disconnected users are logged and later propagated to servers. Several problems may arise for such updates:

(1) conflicting updates from different users,

(2) unpredictability of concurrent update results,

(3) inaccessibility of remote non-replicated content.

There are several proposals to achieve a serializable isolation level for mobile transactions, such as the use of speculative locking [18], multiversioning [3] and locking protocols [12] [15]. The strong correctness criterion of one-copy serializability (1CS) [3] needs to be compromised for mobile settings.

Caching of frequently accessed data is crucial for mobile computing since it alleviates availability problems during weakly connected and disconnected periods. Consistent serial caching in mobile nodes is outlined in [10]. Buffering at the base station level using broadcasts instead of cashes has been proposed in [14]. Since a large database typically cannot be cached in its entirety in a small mobile device, transactions need to be very simple; e.g., [17] proposes to realize them by stored procedures.

For many mobile applications, transaction conflicts tend to be rare, if not completely absent, so that strong serializability is not well-suited, particularly whenever Two Phase Locking (2PL) [3] is used. It may cause a transaction to block that is the more annoying the more disconnections occur. In fact, most databases on the market are content to provide snapshot isolation (SI) [2], which reduces blocking and completely avoids it for read operations. Generalized SI (GSI) extends SI to replicated settings [6]. However, transactions may only see a snapshot that is older than the state at their commencement, although the interesting non-blocking properties of SI are preserved. So, the number of message rounds can be significantly reduced: only transactions with write access need to communicate with the base station.

4. Isolation Level Criteria

There are several correctness criteria for transactions. They are based on the one-copy-equivalence principle, i.e., several physical copies of an object viewed as a single logical object. We first outline established criteria for the non-distributed case before moving on to distributed and replicated scenarios.

In general, a transaction contains a set of read and write operations that lead the database from one consistent state to another. A "history" models the interleaved execution of concurrent 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 them is a write operation. The most straightforward solution is to implement serial scheduling by way of two-phase locking (2PL).

Weaker consistency levels have been considered, for reducing the blocking of transactions. Weaker consistency corresponds to a weaker isolation level, such as Snapshot Isolation (SI). In SI, each transaction T reads data from a snapshot of committed data, taken when the transaction started (*Tstart*), which may be any time before the transaction's first read. A transaction running in SI is never blocked for reads from their own snapshot, while seeing their own updates. Updates by other transactions active after *Tstart* are invisible to T. Thus, SI is a kind of multiversion concurrency control. Transactions are committed according to the first-committer-wins rule, with commit timestamps *Tcommit* that are greater than any already existing one. The transaction successfully commits only if there is no other transaction T' with a commit timestamp in the interval [*Tstart*, *Tcommit*] that wrote data that T also wrote. Otherwise, T will abort.

For coping with the need to further relax isolation levels for the distributed and fully replicated setting, One Copy Serializable (1CS), Generalized Snapshot Isolation (GSI) and One Copy Snapshot Isolation (1CSI) schedules have been proposed [6, 11, 8].

4.1. One Copy Serializable (1CS)

1CS is the strongest correctness criterion for replicated databases. Replication is transparent to transactions. Interleaved execution with other transactions is equivalent to a serially ordered execution of the transactions, i.e., all available replicas see the same result.

4.2. Generalized Snapshot Isolation (GSI)

With SI, data are read from a snapshot of data committed at *Tstart*. The principles of SI are postulated in [6], where also GSI as a first attempt to port SI to a replicated setting is addressed. In GSI, a transaction sees a local, possibly not up-to-date snapshot. Snapshots may be older than in a non-distributed database since GSI takes into account that writesets usually cannot be applied in all replicas at a time. Thus, transaction writes may see outdated content (the more so the older the adopted snapshot is), in which case they are eventually aborted by a younger transaction. Yet, all relevant properties of SI still hold for GSI, in particular those that guarantee a serializable behavior. Commitment of read-only transactions is immediate, even for outdated content. For a transaction T with non-empty writeset, GSI requires a certification which dynamically enforces that no other transaction has updated content that T has updated too: The database checks whether T' writeset intersects with the writeset of any other transaction committed after *Tstart*. If all intersections are empty, T commits, else, T aborts.

4.3. One Copy Snapshot Isolation (1CSI)

1CSI names a refinement of GSI where transactions always see up-to-date content, i.e., the latest snapshot. So, 1CSI is an attempt to realize strong SI properties in a replicated setting. Its most straightforward implementation may be accomplished by sending the readsets and writesets of transactions to the DBS and the appropriate snapshot check (similar to the certification process) is checked there [8]. This procedure is not feasible in a mobile environment due to its inherent constraints and even for fixed networks [9]. Another alternative approach is to block the beginning of transactions till the latest snapshot version is gotten which may lead to higher response time for read-only transactions [8]. Furthermore, it will block (at the beginning of the transaction) read operations when they are executed under SI, making one of its most attractive features useless.

5. Implementing GSI

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

5.1. Motivation

We assume that mobile transactions are first executed at the MH and are later validated by the DBS. It is reasonable to assume that MTs prefer committed data, although at first, they must deal with possibly uncommitted data from the BS. According to TPC-W [19], transactions conform to certain patterns of data accesses. Hence, if most operations are read-only, interaction among replicas will be fairly low, i.e., GSI then is most suitable. As an example, consider roam-

ing users interested in, e.g., hotel locations, special events, ticketing information etc.

For write operations, interaction with the DBS is needed for certification. Moreover, transactions on uncommitted data by disconnected users may lead to cascading aborts [3]. On the other hand, MHs working with uncommitted data may receive information of the outcome of an update transaction by short messaging, paging services or other means provided by mobile communication. Thus, GSI is ensured, while minimizing the amount of messages, as long as downloads of accessed database content is non-frequent and message upload is reduced to a few writeset items, if any. That way, the main properties of SI will be preserved, and with appropriate static or dynamic conditions, serializable histories can be generated while the number of messages remains low.

5.2. Transaction Execution

MH starts the execution of an MT by contacting the BS of its covering cell. The BS then returns the requested content, executing the transaction on the snapshot of the database from which the content is taken. The BS maintains a transaction associated to MT in its underlying DBMS, since additional information may have to be retrieved from the same snapshot. However, for simplicity, we assume that at most one message is needed for the BS.

Read-only accesses of an MT can be committed without further ado, while for committing writes, key/value pairs of the MT's writeset, along with the snapshot, are sent to the BS. The BS then sends the message to the DBS which in turn executes the GSI certification process for MT. If it turns out that MT's write data have been updated already before, MT is rolled back and an abort message is sent to the BS. Otherwise, MT is committed, and the DBS sends a commit message (along with all possible missed updates since the snapshot version of MT) to the BS. The BS forwards the commit or abort message to the MH. If committed, the cached version then becomes the committed state. If aborted, nothing happens to transactions executed at MH after or concurrent to MT since they obtain a consistent (probably older) snapshot version.

Note that MH roaming does not affect the transaction management. Forwarding commit or abort messages is affected only if there is a hand-over during the certification process. Moreover, the validation of the transaction may be notified by other means such as using a SMS or MMS if the MH remains disconnected.

Once MT has been committed, the scheduler multicasts the writeset of the certified MT along with its *Tcommit* timestamp to the rest of BS, using a FIFO communication service [5]. In turn, the BSs update their database copy while keeping a consistent older snapshot of the database. That constitutes the key idea about how GSI is guaranteed within this transactional 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.

Several optimizations may be considered. One is that the DBS can be one of the BS itself. Its state can be maintained by a failure detector [4], so that, if BS crashes, one of the remaining BSs can replace it as the new DBS. Moreover, a full replicated certifier could be implemented for the entire system. However, that does not scale well, since all updates would have to be applied at all available BSs. Note that both solutions only may make sense for WLANs, but not for public mobile networks since, in densely populated microcell-covered areas, replication and failure management at this level is not feasible [9].

6. Related Work

We briefly review some existing mobile transaction management solutions. They have been conceived as an extension of distributed transaction management in fixed networks. However, divergent characteristics for fixed and mobile networks have been pointed out in [9], such as bandwidth limitations and frequent disruptions in mobile networks that may need special reconciliation techniques. Subsequently, two mechanisms for achieving serialization by locking are sketched. In [18], a speculative locking mechanism which converts the write lock to a short lock of the 2PL is described. A write lock is released as soon as its afterimage is generated, under the assumption that there are no user aborts. Hence, read-only transactions may read from two versions. However, if an update transaction is rolled back, a complex mechanism of cascading aborts is entailed. Moreover, read-only transactions have to wait for the commitment of previous transactions and even of transient values. A modification of 2PL by adding a validate lock which contains a committed but not validated version is proposed in [12]. In both mechanisms, timestamps (using version numbers) are used to prevent deadlocks. Transactions are executed as stored procedures where MTs must wait for acquiring all proper locks before issuing the operations at the MH. Our approach is totally different: it exchanges fewer messages with the DBS and, thanks to GSI, avoids all of the blocking time that otherwise is wasted at the DBS. Multiversion reconciliation for providing serializability is introduced in [15]: serializability is ensured, although conflict reconciliation is left to the application.

Driven by burgeoning market needs, commercial vendors are beginning to offer solutions for mobile databases. The one of Oracle Lite 10g [13] is similar to ours. The main difference is the way snapshots are handled: Oracle needs different snapshots for different purposes (read-

only, updatable, etc), whereas we provide a, possibly, older database snapshot of the information needed by the MH. Moreover, our certification-based conflict detection is application-independent and does not rely on the DBS. Currently, a growing trend toward middleware support [1] and service-oriented architectures for mobile databases [7] can be observed.

7. Conclusion

We have adapted the GSI transaction correctness criterion of distributed databases to be usable for replicated mobile networks. Our adaptation of GSI is especially useful for mobile settings since the number of exchanged messages per transaction is significantly reduced. Thus, many limitations associated with mobile devices can be overcome. In particular, dynamic web content generation for emerging mobile applications is expected to work very well.

References

[1] M. Beigl. MODBC - a middleware for accessing databases from mobile computers. In *3rd Cabernet Plenary Workshop*, 1997.

[2] H. Berenson, P. A. Bernstein, J. Gray, J. Melton, E. J. O'Neil, and P. E. O'Neil. A critique of ansi sql isolation levels. In *SIGMOD Conference*, pages 1–10, 1995.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

[4] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[5] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[6] S. Elnikety, F. Pedone, and W. Zwaenopoel. Database replication using generalized snapshot isolation. In *SRDS*. IEEE-CS, 2005.

[7] C. Gollmick. Client-oriented replication in mobile database environments. Technical Report MINET 03-08, Univ. of Jena, 2003.

[8] J. R. González de Mendívil, J. E. Armendáriz-Íñigo, J. R. Garitagoitia, L. Irún-Briz, and F. D. Muñoz-Escoí. Non-blocking ROWA protocols implement GSI using SI replicas. Technical Report ITI-ITE-06/04, Instituto Tecnológico de Informática, 2006.

[9] J. Gray, P. Helland, P. E. O'Neil, and D. Shasha. The dangers of replication and a solution. In H. V. Jagadish and I. S. Mumick, editors, *SIGMOD Conference*, pages 173–182. ACM Press, 1996.

[10] S. Lee, C.-S. Hwang, and H. Yu. Supporting transactional cache consistency in mobile database systems. In *MobiDe*, pages 6–13, New York, NY, USA, 1999. ACM Press.

[11] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *SIGMOD Conference*, 2005.

[12] S. K. Madria, M. Baseer, and S. S. Bhowmick. A multiversion transaction model to improve data availability in mobile computing. In *CoopIS/DOA/ODBASE*, volume 2519 of *LNCS*, pages 322–338. Springer, 2002.

[13] Oracle. Oracle database lite. Accessible in URL: `http://www.oracle.com/technology/products/lite/index.html`, 2007.

[14] W.-C. Peng and M.-S. Chen. Query processing in a mobile computing environment: Exploiting the features of asymmetry. *IEEE TKDE*, 17(7):982–996, 2005.

[15] S. H. Phatak and B. R. Badrinath. Multiversion reconciliation for mobile databases. In *ICDE*, pages 582–589. IEEE-CS, 1999.

[16] C. Plattner, G. Alonso, and M. Tamer-Özsu. Extending DBMSs with satellite databases. *VLDB J.*, 2006. *Accepted for publication*.

[17] N. M. Preguiça, C. Baquero, F. Moura, J. L. Martins, R. C. Oliveira, H. J. L. Domingos, J. O. Pereira, and S. Duarte. Mobile transaction management in mobisnap. In *ADBIS-DASFAA*, volume 1884 of *LNCS*, pages 379–386. Springer, 2000.

[18] P. K. Reddy and M. Kitsuregawa. Speculative locking protocols to improve performance for distributed database system. *IEEE TKDE*, 16(2):154–169, 2004.

[19] TPC-W. Transaction processing performance council. Accessible in URL: `http://www.tpc.org`, 2007.