SIPRe: A Partial Database Replication Protocol with SI Replicas

J. E. Armendáriz, A. Mauch, J. R. González de Mendívil, F. D. Muñoz

Depto. de Ing. Matemática e Informática - Univ. Pública de Navarra
Campus de Arrosadía, 31006 Pamplona, Spain

Instituto Tecnológico de Informática - Univ. Politécnica de Valencia
Camino de Vera, s/n - 46022 Valencia, Spain

{enrique.armendariz,augusto.mauch,mendivil}@unavarra.es, fmunyoz@iti.upv.es

Technical Report TR-ITI-ITE-07/21

SIPRe: A Partial Database Replication Protocol with SI Replicas

J. E. Armendáriz, A. Mauch, J. R. González de Mendívil, F. D. Muñoz

Depto. de Ing. Matemática e Informática - Univ. Pública de Navarra
Campus de Arrosadía, 31006 Pamplona, Spain

Instituto Tecnológico de Informática - Univ. Politécnica de Valencia
Camino de Vera, s/n - 46022 Valencia, Spain

Technical Report TR-ITI-ITE-07/21

e-mail: {enrique.armendariz,augusto.mauch,mendivil}@unavarra.es,
fmunyoz@iti.upv.es

October 23, 2007

Abstract

Database replication has been researched as a solution to overcome the problems of performance and availability of distributed systems. Full database replication, based on group communication systems, is an attempt to enhance performance that works well for a reduced number of sites. If application locality is taken into consideration, partial replication, i.e. not all sites store the full database, also enhances scalability. On the other hand, it is needed to keep all copies consistent. If each DBMS provides SI, the execution of transactions has to be coordinated so as to obtain Generalized-SI (GSI). In this paper, a partial replication protocol providing GSI is introduced that gives a consistent view of the database, providing an adaptive replication technique and supporting the failure and recovery of replicas.

1 Introduction

Distributed databases have become an attractive approach for providing service to large number of users. Data is stored at multiple sites geographically distributed. Under these systems, database replication is one attractive approach. On one hand, it increases data availability in the presence of failures. On the other hand, it can improve the system performance, specially in retrieval costs by exploiting local access. The management of replicated data, i.e. the decision of when, how and where to perform updates [21, 12, 20], to maintain data consistency may have a non-negligible overhead in the system too.

Full database replication (all sites, or replicas, store a copy of the database) using the eager and update everywhere approach has been shown as an attractive way to perform replication [12]. Among all replication techniques under this assumption, those based on Group Communication Systems (GCS) [5] are the most promising [21].They take advantage of order and atomicity properties provided by the GCS. These GCS-based solutions perform as follows: each transaction is locally executed at a single replica (its delegate replica); once the commit of this transaction is requested, updates are collected (denoted as writeset) and multicast, using the total order primitive [5]. Upon the delivery of this message the outcome of the transaction can be determined in different manners, see [21]. This paper focuses in the certification-based technique in which each site autonomously determines if the delivered transaction should be aborted or committed. A transaction can be committed if it does not *conflict* with any concurrent, though previously committed, transaction in the system. To carry out this task, each replica holds a log of previously certified transactions to compare with new ones.

Earlier solutions [15] tried to provide the strongest correctness criterion which is 1-Copy-Serializability (1CS) [4] that limits concurrency and, thus, scalability. Moreover, most commercial databases provide Snapshot Isolation (SI) [3], where read operations never block since they read from the latest committed version of the database at the moment it started, and several works have dealt with providing a 1-copy equivalence for SI replicas which will be called Generalized-SI (GSI) [9][11] throughout this work. It differs from centralized SI in that transactions may read from an older committed version of the database as opposed to SI; however, all the interesting properties of SI remain the same.

Nevertheless, all these GCS-based solutions lack of scalability as it has been shown in [14] since all updates have to be applied at all replicas to keep them consistent. This fact implies that each replica has a portion of its scheduling devoted to apply updates from other replicas and this fraction can become quite large, i.e. the addition of new replicas adds more overhead instead of alleviating this effect. At this point, it is where partial replication makes sense. It consists in splitting the database in portions (partitions) according to application semantics, and then by replicating each fragment at a subset of available replicas [18, 13, 17, 6]. This paper proposes an algorithm for partial replication, called SIPRe that provides GSI. It supports the execution of distributed transactions where some operations of a transaction can be submitted to different replicas since partitioning is not perfectly done. An outline of its correctness is also given. Afterwards, following the steps presented in [13], an adaptive replication schema is proposed. Adaptive replication means that the replication pattern for a data item may change as changes occur in the read-write access pattern at different sites. A study of the fault tolerance and recovery issues is also given.

The rest of this paper is organized as follows: Section 2 presents an abstraction of the system model used. SIPRe is described, along with its correctness, in Section 3. The partitioning schema can be dynamically modified at any moment and it is shown in Section 4. The crash and recovery issues are outlined in Section 5. A review of previous related works is done in Section 6. Finally, conclusions end the paper.

2 System Model Overview

The system assumed in this paper is an abstraction of a middleware database replication architecture. The system is composed by n sites (or nodes) and a database (DB) composed by a set of finite data items. Each site runs a SIPRe instance that coordinates its execution by message exchange thanks to the usage of a a Group Communication System (GCS) [5]. This GCS features a total-order [5] and a reliable multicast/unicast as communication primitives of the Communication Service (CS) as well as a membership service (MS) with virtual synchrony [5] and preventing the contamination phenomenon [8]. Besides, each site R_k contains a Database Management System (DBMS) maintaining a copy of a subset of the items in the database (none stores a full copy of the database), $Items(\mathsf{R}_k) \subseteq \mathsf{DB}$, and providing SI [3] to transactions locally executed. Clients access to the system by way of a standard interface, e.g. JDBCto issue transactions. A transaction T represents a sequence of read and write operations followed by a commit or abort operation.

3 Protocol Description

Figure 1 shows the SIPRe replication protocol as an event based system that we explain in the following. First of all, it is very important to fix the transaction master site (or delegate replica, $T.master$). This is not an intuitive task, though not overwhelming, since it highly depends on the kind of application considered. Taking into consideration the TPC-W benchmark [19], the transaction execution flow is fixed (which also may serve to partition the database, though the partitioning process itself is not discussed in this paper), i.e. the probabilities that a transaction may switch from one table to another are defined. In the following, it is assumed that the transaction (before its begin operation) has been transparently forwarded to its master site. Hence, there exists a *lookup* service to forward the transaction to the most appropriate site.

During its read and write operations, it can potentially access data that belongs to another replica. It is needed that T "*sees*" the same snapshot throughout all replicas it may potentially need for its execution. In order to achieve this, a start message is multicast using the total order service, step I.1, at the beginning of the transaction.

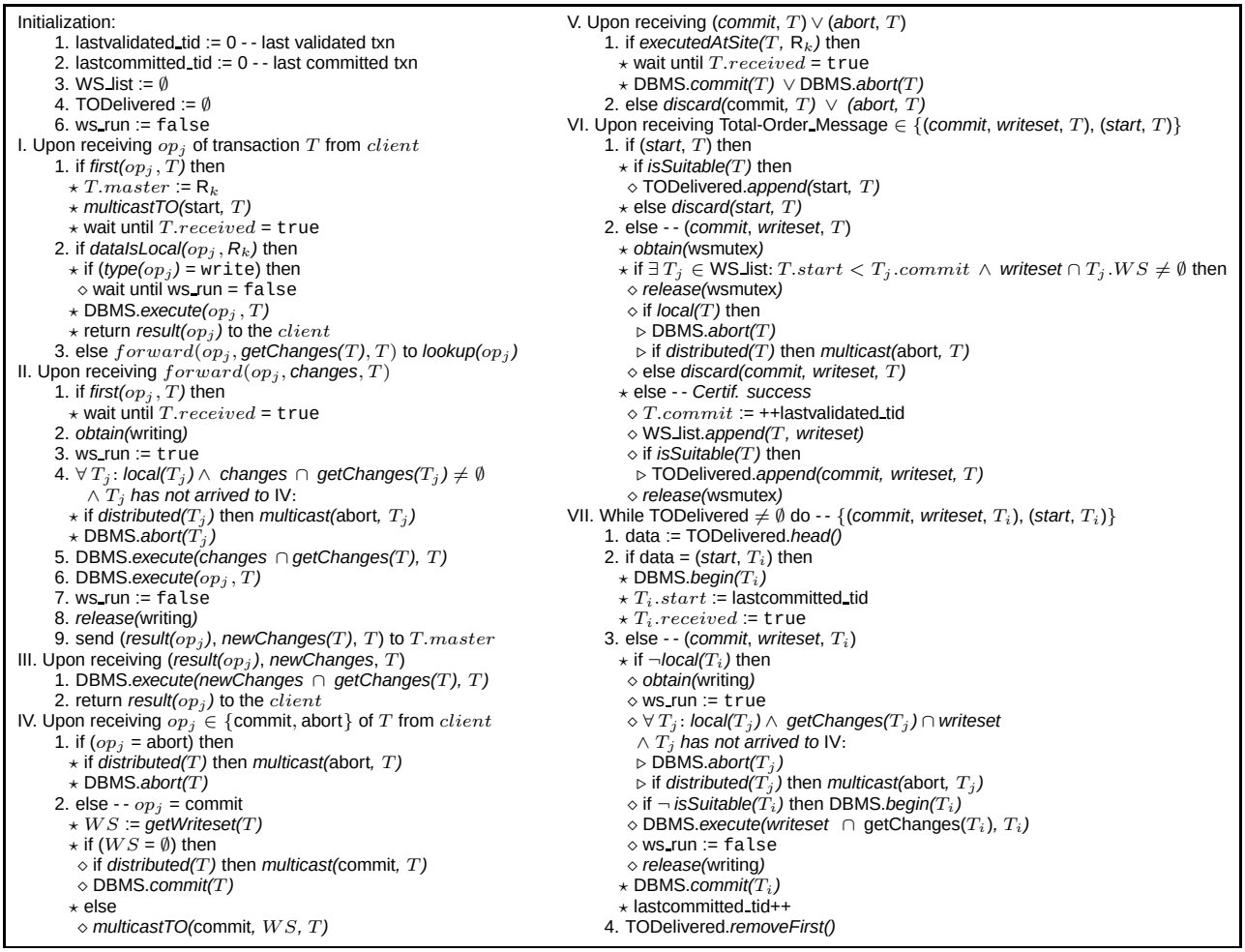

Figure 1: SI Partial Replication (SI-PRe) protocol at replica R_k

Upon the delivery of this message (step VI.I), the proper set of replicas, which is delimited by the isSuitable() function uniquely determines for each T and its $T.master$ a single site for each partition it may potentially need (this can be inferred by the *lookup* service at each replica), enqueue the message at them in the TODelivered variable to start the transaction. This last variable is globally total-ordered at each replica that permits to commit transactions in the very same order and to obtain the same snapshot at all replicas potentially needed ($T.start$), step VII.2. From this moment on, the transaction can start the execution of read and write operations. At some point during its execution, it might happen that a given statement could not be executed at $T.master$ and the operation should be forwarded to another replica containing the information. Moreover, it can be the case of a statement, op_j, that depends on its own previously updates and these changes must be also propagated along with the respective statement to be executed (step I.3) in a $forward$ message to the *lookup(op_j)* replica.

Once the $forward$ message is delivered to the proper replica is important that changes brought by the transaction are properly applied, i.e. T must not be aborted by local transactions (step II). All local conflicting transactions are aborted, the operation is executed and changes are sent back to $T.master$. Furthermore, no write operation is allowed until the changes and the operation is done, thanks to the ws_run variable. From the explanation outlined, it should be clear that a transaction can only be aborted at its delegate replica and, hence, the number of messages per transaction is greatly reduced thanks to this. The reception of the previous remote operation (step III) at $T.master$ implies that the result must be returned to the client and, if necessary, changes modifying the state of the transaction should be also applied (e.g. an update operation executed at another replica).

Once all operations of the transaction have been done, the client requests the commit of transaction T (step IV); for the sake of clarity, client explicit aborts are not considered. All updates performed are grouped to form a writeset (*getWriteset(T)* function). If the writeset is empty then the transaction will be straightly committed, in the case of affecting other replicas a commit message is multicast using the basic service that will commit the transaction at these replicas (step V). Otherwise, the writeset is multicast using the total order facility. When a replica receives a writeset (step VI.2), it applies the certification test, no matter which partition it stores, and checks the associated *writeset* field of T against the writesets of its overlapping transactions T_j ($T.start < T_j.commit \wedge$ *writeset* $\cap T_j.WS \neq \emptyset$) contained in the WS_list. If the certification is passed, T will be appended to WS_list queue and its $T.commit$ field gets updated to the auto-incremented value of lastvalidated_tid. Finally, if the replica holds a partition affected by the validated writeset, it will be stored in the TODelivered queue to be applied and committed in the replica.

The last part of the algorithm (step VII.3) governs the application of validated writesets in the system in the proper set of replicas, i.e. those already holding a copy of the validated writeset. These writesets are sequentially applied, i.e. another validated writeset cannot be started until the previous validated writeset has been applied and committed. Again, this fact does not prevent conflict appearance with current executing local transactions. These conflict checks can be done in an automatized way. Furthermore, new write operations are not allowed in the replica so that the writeset will be eventually applied and committed.

3.1 Outline of Its Correctness

We need to show that each set of replicated partitions behaves in the same way, and, since there is no overlapping between partitions, the compound view of partitions corresponds to the global state of the replicated database. In the following, the safety and liveness criteria that SIPRe must satisfy are going to be introduced. The safety criterion establishes that for any pair of sites storing the same partition the *log* of committed transactions in their respective database systems is either the prefix of the other or vice versa. As database systems provide SI, this criterion implies that each database replica at every site holding the same partition has installed the same snapshots in the very same order. Therefore, each database partition reaches the same state, at commit time, for every executed transaction. The liveness criterion must ensure that if a site commits a transaction, it will be eventually committed at every correct replica. Under this criterion, all correct and available databases do not lose any committed update in any database of the system.

The safety criterion is also very important to determine the final isolation level achieved by the SIPRe. It has been stated at the beginning of this work that SIPRe provides GSI [9]. Actually, GSI is an extension of SI best suited for replicated environments. The GSI level allows the use of *older* database snapshots, facilitating its replicated implementation. A transaction may receive a snapshot that happened in the system before its first operation (instead of its current snapshot as in SI). To commit a transaction it is necessary, as in SI, that no other update operation of recently committed transactions conflicts with its update operations. Thus, a transaction can observe an older snapshot but its write operations are still valid update operations for the database at commit time. Many of the desirable properties of SI remain also in GSI, in particular, read-only transactions never became blocked and neither they cause update transaction to block or abort.

In the following, it is shown how the previous concept of GSI level is applied to SIPRe. Suppose that a transaction T starts its execution at its delegate replica ($T.master = \mathsf{R}_k$), it will not be allowed to execute any operation until its associated $start$ message is received. Furthermore, all replicas that can potentially be accessed by T ($\{\mathsf{R}_{k'}$: *isSuitable(T)*$\}$) will obtain the same snapshot. However, this fact does not prevent from getting an *older* snapshot. Suppose that no update operations have occurred at the master site of T, i.e. its lastcommitted_tid $= 0$. Prior to this, a transaction T' belonging to the same partition, whose master is $\mathsf{R}_{k'} \neq \mathsf{R}_k$, has been certified and committed at all available replicas such that *isSuitable(T')* but R_k, due to, e.g., a communication delay in the propagation of messages by the GCS to R_k. Assume that the associated writeset of T contains data item X and, hence, a new version of this item ($X_{T'}$) has been installed in the system. If transaction T reads data item X it will read the version X_0 instead of the already committed and installed version in the system. Moreover, if a transaction T'' concurrently starts at R_k, though after T' has been committed, it will read version $X_{T'}$, since transactions firstly perform their operations at their master replicas and SIPRe validates transactions at all replicas in the same total order. This validation order governs the sequential commit of transactions for their respective associated partition and, hence, it satisfies the safety criterion for the executed transactions which is a sufficient condition for obtaining GSI

level for committed transactions [11].

4 Adaptiveness Issues

From all the topics covered right now about partial replication, it is clear that it is very important to know the application access patterns and to properly partition the database. As it has been pointed out, the less number of message exchanges per transaction the better a replication protocol works [12]. In general, the possible transitions for an application, i.e. the set of tables accessed by a transaction, is well-known in advance, e.g. TPC-W [19]. However, this does not fully avoid the need of forwarding operations to other replicas. Taking this to the extreme, the most straightforward solution is to fully replicate the system but this does not scale well [14, 17].

It is also worth noting the problem of remote access to other replicas due to the application access to several partitions located at different replicas [13]. In SIPRe it has been circumvented by way of the start message at the beginning of the transaction and, hence, every accessed replica obtains the same snapshot. However, it may occur that the partition is far away from perfect and operations at other replicas may happen. Hence, once objects are retrieved at the master replica, they can be temporarily maintained at the replica in memory. This temporary database is a cache of non-local data items that can be perfectly valid for transactions executed at the replica while there are no updates of these items which are outdated by the given snapshot version in the replica. This is easy to check in SIPRe since all replicas validate and certify all transactions so the cached items can be purged once a validated transaction has updated any of them. Of course, items can be removed from the temporary database using a LRU or LFU policy. The previous temporary database is taken here as a first step for adaptive replication. Exploiting access locality is a first attempt for partitioning that can be far away from perfect; system or organizational configuration may change and, hence, former seldom accessed data items at a given location may be more accessed. It makes sense to transfer the partition to this location. The number of messages exchanged per transaction at this replica is greatly reduced. The metric used here can be the number of unicast messages sent that belong to a certain non-local partition and the data transfer technique of this partition can be the one depicted in [2].

5 Fault Tolerance Issues

So far, SIPRe has been presented without considering failures. Hereafter, it is assumed a partially synchronous system and a partial amnesia crash [7] failure model as we want to deal with replica recovery. The MS provides strong view synchrony (current set of active and connected replicas) and SIPRe needs a primary component membership [5]. Changes in the composition of replicas fire a view change event and the GCS groups delivered messages in the views they were sent. Finally, the CS provides uniform total order multicast [5] preventing the contamination phenomenon [8].

Each time a replica fails its associated replicated portions must be kept available to maintain the same number of replicated partitions of the database. Upon firing a view change event due to this, a *fictitious* recovery of a replica that does not host those partitions is done; this can be done by the two-stage recovery protocol presented in [2]. Besides, all replicas store updates affecting the crashed replica to be transferred once the crashed replica joins again the system. This can be done at the certification phase, each time a certified writeset updates a portion belonging to the crashed node, it is appended to a queue of missed updates. Another optimization is to store the transaction identifier of the last committed transaction before the crash failure.

Later, the crashed replica will rejoin the system firing again a new view change event. At this moment the replica storing its partitions of the database will transfer the missed data to the recovering replica, starting from its –i.e, that of the recovering node– last committed transaction identifier using the transfer technique commented above.

6 Related Works

Partial data replication has been extensively studied in the context of distributed database system. It can be classified according to the transaction access pattern, i.e. all operations of a transaction are entirely executed at their delegate replica [6, 16, 18] or not [13, 10, 17]. The first group considers that at least one node holds all data the transaction accesses, while the other group permits to handle distributed transactions where different operations of the same transaction are executed at different sites.

Considering the first group of protocols, a replication algorithm is proposed in [6], where databases are replicated in a cluster. Each incoming transaction is submitted, via a load balancer, to the best cluster node, which will be the transaction delegate site. Then, the transaction is associated with a chronological timestamp and is multicast to all other nodes where there is a replica. Transactions must be performed in every node according to their timestamp order. When a transaction arrives at a non-delegate node it is scheduled but not submitted for execution until the reception of the corresponding writeset. At delegate node, after the commitment of a transaction, its writeset is multicast to the other nodes where there is a replica. Upon the reception of this writeset, the content of the transaction is replaced with its writeset and it can be executed according to the chronological order. SIPRe does not need to assign a global chronological timestamp to transactions, since transaction execution order relies on the total-order delivery. Other works [16, 18] extend the Database State Machine approach to partial replication. In [16], the authors propose two algorithms. In the first algorithm transactions are certified sequentially, which can be a problem if many transactions are submitted. The second algorithm solves this problem by allowing a sequence of transactions to be proposed in consensus instances and by changing the certification test accordingly. In [18] transactions are also executed only in its delegate replica. After its execution, the readset and the writeset of the transaction are multicast in total order, so all replicas can run a certification process to decide on the transaction outcome. The algorithms proposed in [16, 18] ensure 1CS by propagating the whole readsets and writesets of transactions. SIPRe provides GSI, so there is no need to propagate readsets of transactions.

All the algorithms described so far suppose the existence of a perfect fragmentation of the database, so transactions can be executed completely in its delegate site. This database perfect fragmentation is very difficult to achieve [13]. SIPRe does not make that assumption and permits the distributed processing of transactions among a set of replicas, and, just in case, provides adaptive replication.

The second group of replication protocols does not suppose either a database perfect fragmentation. The idea of adaptive replication used in SIPRe is taken from the algorithm proposed in [13]. In that paper, authors propose an epidemic protocol for partially replicated databases in a WAN environment. Each data item is stored permanently in one o more sites, and other sites may have a temporary cached copy. If a transaction needs an item that is not stored in the site where it is being executed, the site sends a message to one of the sites that stores a permanent copy of the item requesting a copy, along with the associated lock table information. Readsets and writesets are propagated among all the sites to ensure 1CS. SIPRe does not need to use epidemic communication. The algorithm proposed in [10] uses group communication primitives to immediately broadcast read operations to all replicas of an item. All write operations of a transaction are broadcast along with the transaction commit request. Then, an atomic commit protocol ensures transaction atomicity. As opposite to SIPRe that does not require to broadcast the read operations. SIPRe is also based in the algorithm presented in [17]. The main difference is how transactions obtain the same snapshot at all possible replicas they may access. In [17] is supposed the existence of the *getDummyTransaction()* function, which allows to get the proper snapshot of a transaction. The algorithm presented in the current paper proposes a more realistic way to ensure that every operation of a transaction is executed under the same snapshot. Besides, the SIPRe algorithm uses adaptive replication in order to minimize the number of messages exchanged per transaction.

7 Conclusions

It has been presented a partial adaptive replication algorithm (SIPRe) that tries to overcome the overhead imposed by full replication: no replica stores a full copy of the database, only portions of it; and, if the access pattern in a given replica changes, SIPRe permits to dynamically allocate new partitions on that

replica. It allows more concurrency to transactions since it provides GSI [9]. SIPRe needs distributed transactions when the items to be accessed are located in multiple replicas, but it ensures consistent snapshots in such case by way of its start message.

References

[1] *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES 2006, The International Dependability Conference - Bridging Theory and Practice, April 20-22 2006, Vienna University of Technology, Austria*. IEEE-CS, 2006.

[2] José Enrique Armendáriz-Iñigo, Francesc Daniel Muñoz-Escoí, José Ramón Juárez-Rodríguez, José Ramón González de Mendívil, and Bettina Kemme. Evaluating certification protocols in the partial database state machine. In *ARES* [1], pages 855–863.

[3] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD Conf*, pages 1–10, 1995.

[4] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

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

[6] Cedric Coulon, Esther Pacitti, and Patrick Valduriez. Consistency management for partial replication in a high performance database cluster. In *ICPADS*, pages 809–815, Washington, DC, USA, 2005. IEEE-CS.

[7] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78, 1991.

[8] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, 2004.

[9] Sameh Elnikety, Willy Zwaenepoel, and Fernando Pedone. Database replication using generalized snapshot isolation. In *SRDS*, pages 73–84. IEEE-CS, 2005.

[10] Udo Fritzke Jr and Philippe Ingels. Transactions on partially replicated data based on reliable and atomic multicasts. In *ICDCS*, pages 284–291, 2001.

[11] J. R. González de Mendívil, J. E. Armendáriz, F. D. Muñoz, L. Irún, J. R. Garitagoitia, and J. R. Juárez. Non-blocking ROWA protocols implement GSI using SI replicas. Technical Report ITI-ITE-07/10, 2007.

[12] Jim Gray, Pat Helland, Patrick E. O'Neil, and Dennis Shasha. The dangers of replication and a solution. In *SIGMOD Conf.*, pages 173–182, 1996.

[13] JoAnne Holliday, Divyakant Agrawal, and Amr El Abbadi. Partial database replication using epidemic communication. In *ICDCS*, pages 485–493, 2002.

[14] Ricardo Jiménez, Marta Patiño, Gustavo Alonso, and Bettina Kemme. Are quorums an alternative for data replication? *ACM Trans. Database Syst.*, 28(3):257–294, 2003.

[15] Bettina Kemme and Gustavo Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, 2000.

[16] Nicolas Schiper, Rodrigo Schmidt, and Fernando Pedone. Optimistic algorithms for partial database replication. In *OPODIS*, pages 81–93, 2006.

[17] Damián Serrano, Marta Patiño, Ricardo Jiménez, and Bettina Kemme. Boosting database replication scalability through partial replication and 1-copy-SI. In *IEEE PRDC*, pages 129–142, 2007.

[18] A. Sousa, Afrânio Correia Jr., Francisco Moura, José Pereira, and Rui Carlos Oliveira. Evaluating certification protocols in the partial database state machine. In *ARES* [1], pages 855–863.

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

[20] M. Wiesmann, A. Schiper, F. Pedone, B. Kemme, and G. Alonso. Database replication techniques: A three parameter classification. In *SRDS*, pages 206–217, 2000.

[21] Matthias Wiesmann and André Schiper. Comparison of database replication techniques based on total order broadcast. *IEEE TKDE*, 17(4):551–566, April 2005.