# A Middleware Database Replication Protocol Providing Different Isolation Levels

J.R. Juárez*, J.R. González de Mendívil*, J.R. Garitagoitia*, J.E. Armendáriz†, F.D. Muñoz-Escoí†
*Dpto. Matemática e Informática, Universidad Pública de Navarra, 31006 Pamplona, Spain
†Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, 46022 Valencia, Spain
Email: {jr.juarez, mendivil,joserra}@unavarra.es, {armendariz, fmunyoz}@iti.upv.es

***Abstract*— Database replication protocols have been usually designed in order to support a single isolation level. This paper proposes a middleware protocol able to manage three different isolation levels over multi-version DBMSs: GSI, SI, and serializable. This ensures a better support for applications that demand different isolation levels for their transactions. Additionally, this protocol is also able to merge the coordination of the replicas for each isolation level, using a weak voting approach for all of them, whilst other recent protocols need a certifying technique for GSI and SI, or a 2PC rule for serializable level.**

## I. Introduction

Database replication is a very attractive approach since it increases the performance and availability, by storing copies of the same data at multiple sites. The price to pay is to keep copies consistent across the whole system. Several correctness criteria have been developed for replicated databases: One Copy Serializable (1CS) [1]; Generalized Snapshot Isolation (GSI) [2]; and, One Copy Snapshot Isolation (1CSI) [3]. The fulfillment of these correctness criteria depend on the DBMS replicas used at each site.

The implementation of database replication systems has two main approaches. Originally, the DBMS-core was modified so as to include some communication support and means to deal with transactions coming from remote sites [1], [4], [5]. This solution is highly dependent on the DBMS used. The alternative approach is to develop a third software layer that isolates the DBMS details from the replication management [6], [7]. However, middleware solutions often lack scalability and exhibit a number of consistency and performance issues. The reason is that in most cases the middleware has to handle the database as a black box, and hence, cannot take advantage of the many optimizations implemented in the database kernel.

We propose a database replication protocol for a middleware architecture (*Mid-Rep*). In this protocol, Snapshot Isolation SI) [8] database replicas are considered, since most commercial DBMSs provide this isolation level. Most protocols are only able to provide a single isolation level. However, our replication protocol offers much more flexibility to applications, providing different isolation levels to transactions: GSI, SI and serializable (SER). Generally, the different levels featured depend on: the transaction isolation level provided by the underlying DBMS (in this case SI); the ordering of commit operations at all nodes; and, the starting point of transactions [9]. *Mid-Rep* is a non-certified replication protocol which is, up to our knowledge, the first protocol proposed in this way.

## II. The Middleware Approach

In this work we took the advantage from our previous works [7] and other middleware architectures providing database replication [6], [3]. In the following we highlight some aspects dealing with the design of the system and the isolation levels.

While developing replication protocols, it is a must not to re-implement features provided by the underlying DBMS, since DBMSs perform these tasks much more efficiently. The writeset must be efficiently picked up from the DBMS. The writeset contains the updated/inserted/removed tuples identified through the primary key. Furthermore, in order to circumvent the reattempts of writesets proposed in [3], it is needed to provide a conflict detection mechanism [10], which uses the concurrency control support of the underlying DBMS. As shown in [11], it is better to deploy replication protocols with constant interaction. Besides, the update propagation should be made using the total order multicast facility [12] in order to avoid the drawbacks of 2PC protocols [1].

Regarding to the isolation level of the local replica it may (or not) determine the isolation level of the one copy equivalent isolation level reached in a replicated setting. For instance, if a strict serializable DBMS (such as 2PL) is used then a 1CS will be obtained. In order to increase the replication protocol performance, the correctness criterion can be relaxed. Assuming that SI DBMSs are used, we can reach easily GSI using a simple certification process.

Database replication protocols providing GSI or 1CSI [2], [3] are based on a *certification* process to commit a transaction in the system. Thus, it is only needed to multicast (using the total-order facility) [12] one message and keep a log, as part of the replication protocol, of already committed transactions. However, this certification mechanism, which can only provide GSI, has several drawbacks. A garbage collector is needed for keeping track of its log. Besides, to provide serizable executions transaction readsets must be propagated and this is prohibitive.

Nevertheless, achieving a 1CSI isolation level implies blocking transactions at their beginning [9]. However, our protocol does not need the use of certification, hence there is no need of using a garbage collector. For the same reason, it is not

· Upon start request of $T_i$ at site $k$
I. Case $T_i.level = SI$:
1. TO-broadcast($T_i$, $k$, $start$);
· Upon select, update, insert, delete request of $T_i$ at site $k$
I. If first operation of $T_i \wedge T_i.level = SI$ then
1. Await status($T_i$)= $start$;
· Upon commit request of $T_i$ at site $k$
I. $T_i$.WS := getwriteset($T_{ik}$) from local $R_k$;
II. status($T_i$) := $pre\_commit$;
III. TO-broadcast($T_i$, $T_i$.WS);
· Upon TO-deliver($m$) at site $k$
I. Case $m = \langle T_i, node, start\rangle$:
1. If $k = node$ then status($T_i$):= $start$;
2. Else *Ignore message*
II. Case $m = \langle T_i, WS\rangle$:
1. Obtain $wsmutex$
2. If status($T_i$) $\in \{toabort, aborted\}$ then *Ignore message*
3. Else
a. If $T_i$.site = $k$ then
- status($T_i$):= $committable$;
- R-broadcast($\langle T_i, commit\rangle$);
- Await status($T_i$)= $tocommit$;
- $DB^k$.commit($T_i$); status($T_i$):= $committed$;
b. Else
- For all $T_m$ in get_conflicts(WS, $T_i.level$)
$\wedge$ status($T_m$) $\neq committable$ do
$\star$ $DB^k$.abort($T_m$);
$\star$ If status($T_m$) = $pre\_commit$ then R-broadcast($\langle T_i, abort\rangle$);
$\star$ status($T_m$):= $aborted$;
- $DB^k$.apply($T_i$, WS);
- Await status($T_i$) $\in \{tocommit, toabort\}$;
- If status($T_i$) = $tocommit$ then
$\star$ $DB^k$.commit($T_i$); status($T_i$):= $committed$;
- Else
$\star$ $DB^k$.abort($T_i$); status($T_i$):= $aborted$;
4. Release $wsmutex$;
· Upon R-deliver($m$) at site $k$
I. Case $m = \langle T_i, commit\rangle$: status($T_i$):= $tocommit$;
II. Case $m = \langle T_i, abort\rangle$: status($T_i$):= $toabort$;

Fig. 1. *Mid-Rep* algorithm at replica $R_k$

necessary to propagate the readsets to provide serial execution, as needed when using certification.

## III. Mid-Rep Replication Protocol

We propose a non-certified replication protocol (see Figure 1), called *Mid-Rep*, where transactions can select their transaction isolation level (GSI, SI or SER), to be used in a middleware architecture where each node contains a DBMS providing SI transaction isolation level.

Different transactions may be created in the replicated system. Each transaction can select an isolation level, depending on its requirements, at the beginning of its execution. *Mid-Rep* protocol is able to provide GSI level, given that transactions are atomically committed at all nodes and their commit is totally ordered [9].

In order to obtain a serializable level, *Mid-Rep* parses all the read operations for SER transactions to turn them into "`SELECT FOR UPDATE`" statements. This makes possible to see read-write conflicts from the beginning of a transaction till its commit time. Thus, since this satisfies *dynamic serializability condition* (DSC) [2], a serializable level is achieved.

An approximation of the SI level can be easily achieved by using *start* points in the transactions. These *start* points guarantee that, when a transaction begins its execution, it have seen all the changes applied in the system before that point. Thus, to obtain SI, the *Mid-Rep* protocol multicasts a *start* message (using a total order primitive) and the transaction will remain blocked until this message is delivered. Otherwise, the transaction starts straight away its reading and writing phase.

The price to pay for avoiding the read set propagation in the SER mode is awaiting the message reception. Thus, the protocol has two message rounds: a total order message round with the write sets and another reliable message round with the final decision.

During the commit phase, *Mid-Rep* total-order multicasts the write set to all available replicas. Upon delivery of this message at a replica it will block all write operations performed in the replica until the conflict detection and the write set application are done. Moreover, no other write set could be applied until the previous write set application is finished. The write set delivery at the master node will multicast a $commit$ message (if it was not previously aborted). The delivery of this message will commit the application of the write set at the rest of replicas.

## Acknowledgment

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

## References

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

[2] Sameh Elnikety, Fernando Pedone, and Willy Zwaenopoel, "Database replication using generalized snapshot isolation.," in *SRDS*. 2005, IEEE-CS.

[3] Yi Lin, Bettina Kemme, Marta Patiño-Martínez, and Ricardo Jiménez-Peris, "Middleware based data replication providing snapshot isolation.," in *SIGMOD Conference*, 2005.

[4] Michael J. Carey and Miron Livny, "Conflict detection tradeoffs for replicated data.," *ACM Trans. Database Syst.*, vol. 16, no. 4, pp. 703–746, 1991.

[5] Shuqing Wu and Bettina Kemme, "Postgres-R(SI): Combining replica control with concurrency control based on snapshot isolation.," in *ICDE*. 2005, pp. 422–433, IEEE-CS.

[6] Marta Patiño-Martínez, Ricardo Jiménez-Peris, Bettina Kemme, and Gustavo Alonso, "MIDDLE-R: Consistent database replication at the middleware level.," *ACM Trans. Comput. Syst.*, vol. 23, no. 4, pp. 375–423, 2005.

[7] Luis Irún, Hendrik Decker, Rubén de Juan, Francisco Castro, Jose E. Armendáriz, and Francesc D. Muñoz, "MADIS: A slim middleware for database replication.," in *Euro-Par*. 2005, vol. 3648 of *LNCS*, pp. 349–359, Springer.

[8] 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 Conference*. 1995, pp. 1–10, ACM Press.

[9] J. R. González de Mendívil, J. E. Armendáriz, J. R. Garitagoitia, L. Irún, and F. D. Muñoz, "Non-blocking ROWA Protocols Implement GSI Using SI Replicas," Tech. Rep. ITI-ITE-06/04, ITI, 2006.

[10] Francesc D. Muñoz, Jerónimo Pla, María Idoia Ruiz, Luis Irún, Hendrik Decker, José Enrique Armendáriz, and José Ramón González de Mendívil, "Managing transaction conflicts in middleware-based database replication architectures," in *SRDS*. 2006, IEEE-CS, *Accepted for publication*.

[11] Jim Gray, Pat Helland, Patrick E. O'Neil, and Dennis Shasha, "The dangers of replication and a solution.," in *SIGMOD Conference*, 1996, pp. 173–182.

[12] Gregory Chockler, Idit Keidar, and Roman Vitenberg, "Group communication specifications: a comprehensive study.," *ACM Comput. Surv.*, vol. 33, no. 4, pp. 427–469, 2001.