Trying to Cater for Replication Consistency and Integrity of Highly Available Data[*]

J. E. Armendáriz-Iñigo, J. R. Juárez-Rodríguez
Universidad Pública de Navarra,
Pamplona, Spain
{enrique.armendariz, jr.juarez}@unavarra.es

H. Decker, F. D. Muñoz-Escoí
Instituto Tecnológico de Informática,
Valencia, Spain
{hendrik, fmunyoz}@iti.upv.es

Abstract

Availability of databases benefits from distributed replication. Each application and user require different degrees of data availability, consistency and integrity. The latter are perceived as competing objectives that need to be reconciled by appropriate replication strategies. We outline work in progress on a replication middleware architecture for distributed databases. It simultaneously maintains several protocols, so that it can be reconfigured on the fly to the actual needs of availability, consistency and integrity of possibly simultaneous applications and users.

1 Introduction

Because of an increasing demand for on-line data, information service providers are confronted with the need to raise the availability of their offerings. Data replication is emerging as a very promising means of boosting the availability of databases [10, 11]. Replication does not just consist of redundant hardware or backup copies, which are usually off-line, but rather involves a fully transparent on-line distribution of copies of databases or parts thereof, including protocols for replication and failure recovery as well as policies to trade off availability requirements, replica consistency and semantic integrity of data.

Availability of database systems and services in general can significantly benefit from distribution and replication. Applications may take advantage of the higher availability of replicated databases by accessing their *closest* replica to issue transactions. In the same way, whenever a node failure happens, applications accessing this replica will be transparently redirected to alive replicas.

However, three major obstacles need to be overcome when introducing replication for improving the availability of distributed data. The first one is deploying the system itself. There are basically two alternatives: either with DBMS core modifications [15, 21] or by a *middleware* [16, 19, 14, 2]. The former presents a lower overhead but it greatly depends on the DBMS used, whilst the latter gets rid of the DBMS dependency but special attention has to be paid to the introduced overhead.

The second hurdle which may prevent the use of replication is that different users and different services have different requirements on the consistency and availability of data. Hence, in [11] the several replication techniques are classified according to two parameters. The first parameter considers where updates happen: each data item has an owner node that performs all updates and propagates them to the rest of replicas (*primary-copy*); or, updates may occur at any node (*update everywhere*). The second one establishes the coordination among replicas; it may be: *eager* where coordination (mainly, involving update propagation) takes place before the transaction is committed; or, *lazy* coordination occurs after the transaction has been committed. The combination of

[*]We acknowledge support by Spanish MCyT grant TIC2003-09420-C02 and EU grant FP6-2003-IST-2-004152.

these parameters gives a family of replication protocols [15, 19, 21, 16, 1, 12]. Most of them exploit view synchrony and communication primitives as provided by a Group Communication System (GCS) [5], as well some optimization techniques for improving scalability [15, 19]. Replication consistency is achieved either by means of: total-order multicast procedures [15], by priorities [1], or by epidemic algorithms [12].

The third issue which may be a hindrance for deploying replication order to increase availability is the lack of support for maintaining semantic integrity constraints. Centralized databases nowadays offer built-in functionality for checking simple but frequently occurring kinds of integrity constraints. However, this support is lacking for distributed database installations. Usually, the burden of implementing integrity enforcement is passed on to the application programmer or end user. This, nonetheless, typically results in application-dependent, failure-prone code that is hard to maintain and evolve. Thus, it is desirable that the replication architecture itself is providing for some support of integrity enforcement.

In this paper, we outline the intents, purposes and basic ideas of a new middleware architecture for enhancing the availability, consistency and integrity of distributed databases, as part of a planned project called CONFIA (*consistencia y fiabilidad* which, in Spanish, is supposed to mean consistent replication for dependability, subsuming the latter availability and fault tolerance).

Our middleware intends to overcome the three obstacles mentioned above. With regard to the first with standard SQL constructs ready-made SQL functionality and the write ahead log usage the meta data handling for the protocols is accomplished. That way, concurrency control is left to the database while consistency is managed by the replication protocol. They become much less cluttered and thus much easier to develop and implement. This basic feature of CONFIA is an evolution of MADIS [2]. Regarding to failure and recovery management a GCS is used. With regard to the second, CONFIA simultaneously maintains meta data for several protocols, so that the replication strategy can be configured and re-configured seamlessly. Suitable protocols can be chosen, plugged in and exchanged on the fly in order to adapt to the actual needs of given situations. The same may be applied to the recovery process of failed nodes [7]. Solutions for overcoming the third obstacle (the lack of integrity support in replicated databases) are intended to be developed on the basis of an improved theoretical foundation. Currently, perfectly consistent database states are asked for in order to guarantee the correctness of methods for efficient integrity checking. Clearly, this prerequisite is by far too strong for distributed databases with a high volatility of frequently updated data and strong availability requirements, since, in such systems, the data consistency and integrity often is intermittently compromised in favor of keeping the data available. Basic research in CONFIA for improving integrity checking in the presence of inconsistent data aims to remove the annoying intolerance of current foundations.

The rest of the paper is organized as follows: Section 2 addresses the main trends of the CONFIA architecture, emphasizing the exchangeability of protocols for alleviating conflicts between availability and replication consistency. Section 3 discusses how the additional dimension of integrity, which may conflict with both consistency and availability requirements, could be dealt with. Finally, conclusions end the paper.

2 Alleviating Conflicts of Availability and Consistency

The CONFIA architecture is a middleware architecture providing database replication (Figure 1). It is composed by N nodes which communicate among them by message exchange using a GCS. A GCS provides a communication and a membership service [5]. An application submits transactions for its execution in the system. The Database Replication Middleware (DRM) intercepts application requests and applies remote transactions in the DBMS replica. The replication protocol, embedded inside the DRM, coordinates the execution of transactions among different sites to ensure data consistency [4, 9, 16] whilst concurrency is

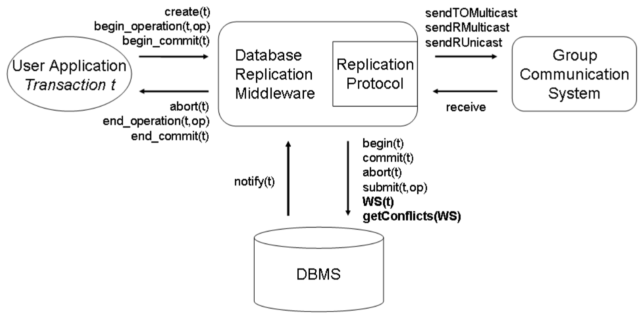

Figure 1. Layered CONFIA architecture.

left to each DBMS replica.

The main goal of this architecture is to keep the protocol overhead to a minimum, i.e. no specific DBMS tasks have to be done by replication protocols, thanks to the automated metadata management performed at the DBMS replica. Moreover, different replication protocols may be plugged in according to the application performance, network configuration (fixed or wireless networks) and some user predefined settings. The same may be applied to the recovery protocol; it is necessary to provide an infrastructure in the DBMS, as it has been done with the replication part, so that the recovery overhead is reduced. In fact, the recovery process is heavier than the replication protocol itself (additional guarantees about uniform message delivery and strong view synchrony).

DBMS. It ensures ACID transactions and complies with a given transaction isolation level [3] to achieve the appropriate consistency criteria [4, 9, 16]. We have added two functions which are not provided by DBMSs, but may be built by standard database functions [2] that are very important from the replication protocol point of view. The first one is $WS(t)$ retrieves the set of objects written by transaction t and its respective log [19]. The other function is $getConflicts(WS)$, as it is interesting to determine the set of conflicting transactions with a remote one. It needs to scan the system's locking tables (e.g., the `PG_LOCKS` in PostgreSQL, the `V$LOCK` view in Oracle 9i, the `DBA_LOCK` in Oracle 10g r2 or the `sys.syslockinfo` table of Microsoft SQL Server 2000) so that this scheme is seamlessly portable to all of them, since only standard SQL constructs are used.

GCS. Group communication systems provide reliable multicast and group membership services. The membership service provides the notion of view, i.e., currently connected and alive nodes. Changes in the composition of a view (addition or deletion) are reported to the recovery protocol. We assume a primary component membership [5], where views installed by all nodes are totally ordered (i.e., there are no concurrent views). The communication system typically features different message delivery guarantees: FIFO, total-order and causal.

Transaction Execution. This depends greatly on the kind of replication considered [11]. In fixed networks, the eager update-everywhere replication technique is the most used one. Transactions are firstly executed at its closest site and updates are grouped and sent to the rest of sites. Once they are (normally total-order) delivered, the commit phase starts, either with some coordination among nodes (such as the 2-Phase-Commit protocol [4]) or with a certification, a test that decides if a transaction may commit or not [4, 9, 16, 15]; the latter does not need to exchange additional messages with other nodes. In lazy schemes, when a conflict arises some reconciliation must be done. This process may be automatically done by the replication protocol or by a notification to the application. In all approaches is necessary either to look for conflicting transactions or re-attempting successfully certified transactions until they got scheduled and committed [16]. The DRM along with the schema modifications and stored procedures of the DBMS help to accomplish this feature [2].

DRM. This is the core of CONFIA and is independent of the underlying database. In [2], we have described a Java implementation, to be used by client applications as a common JDBC driver. It facilitates the plugging and swapping of replication protocols chosen according to given needs and requirements. Protocol swapping, even at runtime, is seamless and fast, since the meta data for each protocol in the CONFIA repertoire (e.g., protocols with eager or lazy update propagation, optimistic or pessimistic concurrency control, etc) is readily at hand at plug-in time.

Orthogonal to this is the performance measurement of CONFIA such a middleware will always tend to be somewhat worse than that of a core-based solution [21]. But the advantage of our middleware solution is to be independent of the underlying database and to be easily portable to other DBMSs. It is easy to show that the performance of the solution will depend on the kind of applications considered as benchmarks, such as TPC-W [20].

3 Distribution and Integrity

In this section, we address some of the complications of integrity checking in a distributed database system (abbr. DDS). In 3.1 we give a brief survey of the state of the art. Then, in 3.2 we sketch some new ideas for approaching integrity checking in DDSs, particularly in replicated ones. Last, in 3.3 we address a fundamental problem of integrity checking in systems where integrity occasionally is compromised in favor of priorities on availability, and indicate solutions.

3.1 State of the Art

In a DDS, the stored data are transparently spread over multiple nodes in a network. (We do not consider non-transparent distribution such as in federated or mirrored databases.) The data in such a system are either fragmented or replicated over several network nodes (mixtures of fragmentation and replication are also conceivable). Fragmentation means that different tables or disjoint sets of rows or columns of database tables (respectively corresponding to horizontal and vertical fragmentation) reside on different nodes. Replication means that multiple copies of a data item exist, one at each node. For simplicity, we only consider homogeneous distribution in this section, i.e., that all network nodes conform to the same underlying database schema.

Apart from questionable recommendations to use triggers for integrity enforcement, the latter is hardly supported at all in current distributed database systems. Many papers on the subject deal with problems of consistency of transactions, concurrency or replication, i.e., the synchronization of the evolution of data at different nodes. But relatively few thorough studies exist for the problem of checking integrity.

Integrity checking can be (and often is) conceived as an application on top of the DBMS. Thus, the simplified evaluation of integrity constraints in distributed databases could in principle take advantage of distribution transparency. However, to implement integrity checking in DDS by transparently running simplified queries could easily induce an unnecessarily high burden of additional network communication and coordination. Also, possibly lots of redundant checking may occur if simplified integrity constraints were evaluated in each network node. Moreover, rolling back updates in the event of integrity violation would incur possibly unaffordable expenses of complex recovery actions at several or all network nodes. Therefore, integrity checking should be an integrated module of the middleware package that drives the data distribution. Knowledge about the given distribution structure can be taken into account for constraint simplification. Clearly, that would not be possible if distribution were transparent to integrity checking.

When data are fragmented over several nodes, the problem is not just to reduce the checking space, i.e., the amount of accessed stored data, as in the non-distributed case. Also the number of nodes to be involved in the evaluation process should be kept as low as possible, as well as the amount of necessary communication and coordination, i.e., the data transferred across the network during integrity checking and maintenance. Each of these three dimensions needs to be minimized for simplifying integrity evaluation.

Each of the mentioned dimensions is minimized in Ibrahim's approach. Roughly, the idea is to first describe the fragmentation of tables, rows and columns by so-called fragmentation rules, i.e., logical expressions which capture the structure of the data distribution. Then, the constraints are rewritten and split up into fragment constraints, in accordance with the fragmentation rules, such that they can be evaluated locally at the nodes where the corresponding data fragments reside, instead of having to evaluate integrity globally. Additionally, simplification methods as described in [18] and others can be applied to the fragment constraints, with regard to a given update or update pattern. An overview of Ibrahim's work can be found in [13].

3.2 Replication and Integrity

In the previous section, replication was not addressed, simply because there is virtually no literature on integrity checking for replicated information. In this section, we describe a first attempt to approach the maintenance of integrity in replicated DDSs.

Evaluating each relevant constraint at each replica would yield an immense amount of redundant checking. Under the assumption that all replica are consistently synchronized, it should suffice to check integrity just in one of them, e.g., in the primary copy, in case there is such a designated node. Synchronization of replica, however, may be eager or lazy, optimistic or pessimistic, or of several other characteristics (cf. [17] for more). Therefore, it is tempting to intertwine a stepwise process of integrity checking with the particularities of the replication protocols at hand, for avoiding additional protocol rounds. But that would entail a very unpleasant dependency of integrity checking from the given protocols and would thus yield insular solutions which lack a sufficient degree of generality and portability.

A solution which is independent of protocols is to wait with integrity checking until the synchronization process has come to a halt, and only then evaluate simplified versions of relevant constraints, at some node. Then, integrity checking can be elegantly parallelized, by assigning the evaluation of different constraints to different nodes which then can work simultaneously and complementarily on the evaluation of integrity. However, this solution is disadvantageous in case of integrity violation, since then, all of the synchronized new states in each replica have to be undone, and a reverse synchronization would have to take place for re-installing the old state.

A solution to the problem of having to undo the new state of all nodes in a replicated DDS in case of integrity violation is to install the new state only in one designated node (ideally, again, a primary copy or master replica), evaluate integrity in that same node and commit the update to be executed in the rest of the network only after integrity has been shown to be satisfied. If integrity is violated, then only the one updated node has to be reset. On the other hand, this solution does not allow the parallelization of integrity evaluation as sketched before. Moreover, doing integrity checking in the new state of one node before replicating this state to all others introduces an additional measure of laziness of replication, which may be unwanted in case the eagerness of replication is required.

Thus the question is: What could constitute a practical compromise between the parallelization of integrity checking by assigning different constraints to different nodes, on one hand, and avoiding laborious rollbacks by doing all of integrity checking at a single node, on the other? In principle, the answer to that question seems to be deceptively simple. If the DDS is not very update-intensive or note very prone to integrity violation but needs to be highly responsive and available also in case of undergoing updates, then parallelization is advisable. If not, then focusing integrity checking on a single site is preferable. Of course, mixed forms of both solutions are easily conceivable. For instance, integrity constraints (which may well be in the hundreds or more) can be assigned to a much more limited number of nodes, each of which could be the designated master for some region of the network.

Integrity checking in a replicated databases is much more complicated, however, in case there is no single designated master copy which is the "owner" of (some region of) the networked database, but different tables (or even different rows or columns of tables) are assigned different owner nodes. Then, the evaluation of some constraint in one node may involve communication with other nodes that are the owners of data to be accessed. In general, the idea is to minimize the amount of additional network communication as much as possible. Hence, the assignment of a constraint I to a particular node should be such that this node can be expected to need less communication with other nodes for evaluating I than others. Fortunately, this probability can in many cases be conveniently determined already at schema specification time, assuming that the ownership of data is specified together with the schema and the specification of the data distribution structure. More generally, the evaluation of integrity constraints can be assigned to nodes also dynamically, which allows one to take into account the given update, the resulting constraint simplifications, the current network load, the present alive state of nodes and other runtime parameters.

None of the ideas for integrity checking in replicated databases as sketched above have been investigated in detail yet, let alone implemented or tried out in practice. So, this section is hardly more than a first outline of a research programme to be pursued in future work. In future work, we hope to report on progress made in this area.

3.3 Solving a Fundamental Problem

Trading off replication consistency and integrity against availability means that, on occasion, it might be necessary to admit updates on database states that are neither guaranteed to be consistent in terms of replication nor in terms of semantic integrity. The tradeoff between availability and replication consistency was already addressed in section 2. To check a given update for integrity in a database state that is not guaranteed to satisfy all constraints, however, is a more fundamental problem: all methods of integrity checking require that the database state before the update satisfies integrity. Only then, it is sufficient to evaluate simplified queries for ensuring integrity of the new state, after the update.

In [6], we have argued that it is possible to safely use well-known query valuation procedures also in the presence of inconsistent data and in particular in the presence of information that does not satisfy integrity. Yet, that did not go as far as to investigate to which extent the correctness of known constraint evaluation methods could still be guaranteed in the presence of already committed data that violate integrity.

Recently, however, we have found that basic theorems about the correctness of tests of well-established integrity checking methods can be significantly relaxed in the following sense: If the integrity test is successful, then all data that have satisfied integrity before the update will continue to satisfy integrity after the update is committed. Thus, this result admits that some of the data in the old state (i.e., before a new update is checked and committed) may have violated integrity, and, under the condition that the method yields a successful test, guarantees that such violations do not infringe the integrity of all data that have been satisfying integrity beforehand. Note that this result has not yet been translated to the distributed (let alone the replicated) case. Therefore, apart from breaching the page limit of this paper, inclusion of a proof of this result would be out of scope. We mention it here only as an outlook into the direction into which our future work is heading. It is expected that the translation of this result to distributed and the replicated databases will provide a firm foundation for our future work on reconciling availability with consistency and integrity in distributed database systems.

4 Conclusion

We have described work in progress on a new replication architecture for information systems. Different applications and users require different kinds of replication strategies with possibly conflicting objectives of availability, consistency and integrity. Hence, a middleware which supports a flexible choice, operation and exchange of suitable protocols is desirable. This innovative kind of flexibility is being realized in CONFIA. We are developing an ample repertoire of replication protocols, each with particular guarantees of consistency traded off with requirements of availability and integrity, from which suitable ones can be chosen, plugged in and exchanged on the fly. We envisage to extend the middleware for use in wireless networks. This requires the development of new protocols which can cope with unstable interconnections, narrower bandwidths, greater heterogeneity of platforms and devices, etc.

Analytical and experimental results, along with some implementation details, appear in [2]. The project DeDiSys [8] currently serves as a testbed for CONFIA.

A major step ahead for enabling a sound approach to integrity checking in replicated databases is expected from the CONFIA project. This is going to take place on the basis of a recent, yet unpublished result. It enables the translation of provably correct technology for efficient integrity checking in non-distributed databases to the replicated case where integrity is partially or intermittently sacrificed in favor of higher replication consistency and availability.

References

[1] J. Armendáriz, J. Juárez, J. Garitagoitia, J. R. G. de Mendívil, and F. Muñoz-Escoí. Implementing database replication protocols based on O2PL in a middleware architecture. In *IASTED DBA*, pages 176–181, 2006.

[2] J. E. Armendáriz, H. Decker, F. D. Muñoz, L. Irún, and R. de Juan. A middleware architecture for supporting adaptable replication of enterprise application data. In *TEAA*, volume 3888 of *LNCS*, pages 29–43. Springer, 2005.

[3] 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. ACM Press, 1995.

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

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

[6] H. Decker. A case for paraconsistent logic as foundation of future information systems. In *CAiSE Workshops (2)*, pages 451–461, 2005.

[7] H. Decker, L. Irún-Briz, F. Castro-Company, F. García-Neiva, and F. D. Muñoz-Escoí. Extending wide-area replication support with mobility and improved recovery. In *ISSADS*, pages 10–20, 2005.

[8] DeDiSys. Dependable distributed systems. Accessible in URL: `http://www.dedysis.org`, 2006.

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

[10] L. Gao, M. Dahlin, A. Nayate, J. Zheng, and A. Iyengar. Improving availability and performance with application-specific data replication. *IEEE Trans. Knowl. Data Eng.*, 17(1):106–120, 2005.

[11] J. Gray, P. Helland, P. E. O'Neil, and D. Shasha. The dangers of replication and a solution. In *SIGMOD Conference*, pages 173–182. ACM Press, 1996.

[12] J. Holliday, R. C. Steinke, D. Agrawal, and A. E. Abbadi. Epidemic algorithms for replicated databases. *IEEE Trans. Knowl. Data Eng.*, 15(5):1218–1238, 2003.

[13] H. Ibrahim. An overview of integrity constraints enforcement for a distributed database. In *PDPTA, Volume II*, pages 822–828. CSREA Press, 2001.

[14] L. Irún, F. Muñoz, H. Decker, and J. M. Bernabéu-Aubán. COPLA: A platform for eager and lazy replication in networked databases. In *ICEIS'03*, volume 1, pages 273–278, Apr. 2003.

[15] B. Kemme. *Database Replication for Clusters of Workstations (ETH Nr. 13864).* PhD thesis, ETHZ, Zurich, Switzerland, 2000.

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

[17] F. D. Muñoz-Escoí, L. Irún-Briz, and H. Decker. Database replication protocols. In L. C. Rivero, J. H. Doorn, and V. E. Ferraggine, editors, *Encyclopedia of Database Technologies and Applications*, pages 153–157. Idea Group, 2005.

[18] J.-M. Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18(3):227–253, 1982.

[19] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. MIDDLE-R: Consistent database replication at the middleware level. *ACM Trans. Comput. Syst.*, 23(4):375–423, 2005.

[20] TPC-W. Transaction processing performance council. Accessible in URL: `http://www.tpc.org`, 2006.

[21] S. Wu and B. Kemme. Postgres-R(SI): Combining replica control with concurrency control based on snapshot isolation. In *ICDE*, pages 422–433. IEEE-CS, 2005.