Toward Reconciling Availability, Consistency and Integrity in Replicated Information Systems

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

J. E. Armendáriz-Ińigo
Universidad Pública de Navarra, Pamplona, Spain
enrique.armendariz@unavarra.es

Abstract

The availability of information systems benefits from distributed replication. Different applications and users 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 architecture for distributed information systems. It simultaneously maintains several protocols, so that it can be re-configured 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 information systems. 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 requirements of availability, replica consistency and semantic integrity of data.

The availability of information systems and services in general can significantly benefit from distribution and replication. Applications that may take advantage of the higher availability of replicated information systems range from frequently updated databases for online transaction processing applications, hourly or daily updated data warehouses, weekly or monthly updated data repositories, read-only services for external users, read- and write-access options for privileged internal users or customers, and many more.

However, three major obstacles need to be overcome when introducing replication for improving the availability of information systems. The first is the error-prone complexity of developing replication protocols for replication and failure recovery, as well as the overhead produced by such protocols for maintaining the consistency of replicated data [2]. For business-critical applications, this can easily amount to a severe impediment.

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. For instance, for the strategic planning of an airline enterprise, the statistics delivered by a data warehouse typically do not rely on most recent updates. Rather, business report services usually refer to selected database states reached cyclically at well-defined breakpoints, without taking into account currently ongoing transactions or intermediate states. On the other hand, information services for flight schedules and reservations need to keep track of up-to-date database states. In general, different classes of users and services typically have different re-
quirements on the accuracy, replication consistency, timeliness and availability of the underlying data. However, in practice, at most just a single manner of replication, a fixed consistency maintenance scheme and a uniform policy for availability and failover management is supported. A more flexible replication architecture which can be adapted to the changing needs of different services and users is therefore desirable.

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. Centralised 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. That, however, 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 information systems, as part of a planned project called CONFIA (consistencia y fiabilidad which, in Spanish, is supposed to mean consistent replication for dependability, the latter subsuming availability and fault tolerance).

Our middleware intends to overcome the three obstacles mentioned above. With regard to the first, standard SQL constructs (views, triggers etc) and ready-made SQL functionality (schema definition, trigger firing etc) are used for implementing major parts of the meta data handling and the network communication of the protocols. That way, the protocols themselves become much less cluttered and thus much easier to develop and implement. This basic feature of CONFIA has to a large extent already been realised in a predecessor project, as described in [7]. 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. 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 far too strong for distributed information systems 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 favour 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.

Section 2 addresses the main features of the CONFIA architecture, emphasising the exchange-ability of protocols for alleviating conflicts between availability and consistency. Section 3 discusses how the additional dimension of integrity, which may conflict with both consistency and availability requirements, could be dealt with. Section 4 concludes the paper.

2 Alleviating Conflicts of Availability and Consistency

The CONFIA architecture is two-layered, as shown in fig. 1. It makes consistency management independent of any DBMS particularities. It takes advantage of ready-made database resources so that protocol overhead is kept to a minimum, as described in [7]. The replication strategy is configurable on the fly, i.e. the architecture allows for plugging in suitable protocols that fit given situations (i.e., the particular needs of different users, use cases and applications, possibly depending on the current network load, device-specific bottlenecks and other operational parameters) best. That way, it is possible to alleviate possibly conflicting priorities of availability and consistency.

The upper layer consists of replication management functions, while the lower one of data-
base schema extensions for storing and processing
transaction data and meta data of protocols for
replication and failure recovery. The upper layer
handles transaction requests from users or appli-
cations and uses meta tables on the lower layer
for transparent replication management. Hence,
the upper layer can be implemented in any pro-
gramming language with an SQL interface. Some
of the meta data tables on the lower layer account
for transactions in the local node (as symbolically
shown in fig. 1), and are updated and maintained
within the transactions accounted for. Communi-
cation between database replicas is controlled by
the consistency manager (CM) which is local to
each network node.

The schema extension also includes some stored
procedures which hide some technical details to
the upper layer. The latter is sandwiched between
client applications and database, acting as a data-
based mediator. Accesses to the database as well
as commit/rollback requests are intercepted, such
that the replication protocol can transparently do
its work. The protocol may access the report ta-
bles to obtain information about transactions, in
order to cater for required consistency guarantees.
The protocol may also manipulate the extended
schema using stored procedures.

The performance of such a middleware will al-
ways tend to be somewhat worse than that of a
core-based solution, which integrates replication
functionality into an existing database kernel im-
plementation such as Postgres-R [9]. But the ad-
vantage of our middleware solution is to be in-
dependent of the underlying database and to be
easily portable to other DBMSs.

As already emphasised, implementation of the
CM, i.e., the core of CONFIA, is independent of
the underlying database. In [8], we have described
a Java implementation, to be used by client ap-
lications as a common JDBC driver. Its consist-
ency control functionality is provided transpar-
ently to users and applications. The CM handles
transaction requests, including multiple sequential
transactions in different JDBC consistency modes,
and communicates with database replicas. It fa-
cilitates the plugging and swapping of replication
protocols chosen according to given needs and re-
quirements. 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.

3 Distribution and Integrity

In this section, we address some of the compli-
cations of integrity checking in a distributed infor-
mation system (abbr. DIS). 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 DISs, particularly in replicated ones.
Last, in 3.3 we address a fundamental problem of
integrity checking in systems where integrity occa-
sionally is compromised in favour of priorities on
availability, and indicate solutions.

3.1 State of the Art

In a DIS, 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 information systems.)
The data in such a system are either fragmented
or replicated over several network nodes (mixtures
of fragmentation and replication are also conceiv-
able). Fragmentation means that different tables
or disjoint sets of rows or columns of database ta-
bles (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 synchronisation 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 information systems could in principle take advantage of distribution transparency. However, to implement integrity checking in DIS 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 minimised for simplifying integrity evaluation.

Each of these dimensions is minimised in Ibrahim’s approach. Roughly, the idea is to first describe the fragmentation of table 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 accordence with the fragmentation rules, such that they can be evaluated locally at the nodes where the corresponding data fragments reside. Additionally, simplification methods as described in [4] and others can be applied to the fragment constraints, with regard to given update patterns. An overview of Ibrahim’s work can be found in [5].

3.2 Replication and Integrity

In section 3.1, replication was not addressed, simply because there is no literature on integrity checking for replicated data. In this section, we describe a first attempt to approach the maintenance of integrity in replicated DISs.

Evaluating each relevant constraint at each replica would yield an immense amount of redundant checking. Under the assumption that all replica are consistently synchronised, 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. Synchronisation of replica, however, may be eager or lazy, optimistic or pessimistic, or of several other characteristics (cf. [6] 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 synchronisation process has come to a halt, and only then evaluate simplified versions of relevant constraints, at some node. Then, integrity checking can be elegantly parallelised, by assigning the evaluation of different constraints to different nodes which then can work simultaneously and complementar-
ily on the evaluation of integrity. However, this solution is disadvantageous in case of integrity violation, since then, all of the synchronised new states in each replica have to be undone, and a reverse synchronisation 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 DIS 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 parallelisation 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 parallelisation 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 DIS is not very update-intensive or not very prone to integrity violation but needs to be highly responsive and available also in case of undergoing updates, then parallelisation 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 information system 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 minimise 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 trade-off 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 [1], we have argued that it is possible to safely use well-known query evaluation 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. We mention it here 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 information 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 realised 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 of a prototype implementation, as well as some implementation details, appear in [7, 8]. The project DeDiSys [3] 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 favour of higher replication consistency and availability.

References