Wide-Area Replication Support for Global Data Repositories *

Hendrik Decker[1], Luis Irún-Briz[1], Rubén de Juan-Marín[1], J. E. Armendáriz[2], Francesc Muñoz-Escoí[1]
[1] Instituto Tecnológico de Informática
Universidad Politécnica de Valencia
Campus de Vera, 46022 Valencia, Spain
{*hendrik, lirun, rjuan, fmunyoz*}*@iti.es*
[2] Dpto. Matemática e Informática
Universidad Pública de Navarra
Campus de Arrosadía, 31006 Pamplona, Spain
enrique.armendariz@unavarra.es

Abstract

Wide-area replication improves the availability and performance of globally distributed data repositories. Protocols needed for maintaining replication consistency may cause an undesirable overhead. Different use cases of repositories suggest the use of different replication protocols, each requiring different meta data. The WADIS middleware for wide-area replication support of distributed data repositories makes use of ready-made database resources such as triggers and views, employing the underlying database management system to support replication protocols, the implementation of which thus becomes more succinct and much simpler. WADIS enables the simultaneous maintenance of multiple meta data for different protocols, so that the latter can be chosen, plugged in and exchanged on the fly in order to adapt to the needs of different use cases best.

1 Introduction

For enhancing the user acceptance of globally distributed data repositories, high availability and fault tolerance are key criteria. Both are supportable by replication. However, a possibly annoying drawback of replication is the overhead for maintaining the consistency of replicated data and the complexity of developing suitable protocols [2].

In this paper, we introduce WADIS, an instance of the MADIS architecture [6] for wide area replication of global repositories. The architecture is two-layered and makes the consistency manager (CM) independent of any DBMS particularities. WADIS takes advantage of ready-made database resources so that the protocols' overhead is kept to a minimum.

The upper layer consists of the protocol functionalities for replication management, the lower of a mechanism for extending the original database schema. The extension exclusively uses standard SQL features such as triggers and stored procedures, so as to provide to the upper layer the information needed to carry out its tasks. Information and meta-data about records created, accessed or deleted in a transaction are automatically stored in particular tables of the extended schema. The consistency protocol thus is able to directly dispose of that information, instead of having to use complex and error-prone routines for that, otherwise. Thus, the handling of each protocol's meta-data becomes much simpler, when compared to other middleware-based systems for replicated databases, e.g., COPLA [3]. The upper layer can be implemented in any programming language, on any platform with an SQL interface, since its functionality exclusively relies on ready-made standard SQL constructs executed by the underlying DBMS. Of course, the performance of such a middleware will always tend to be somewhat worse than that of a core-based solution, such as Postgres-R [4], but its advantage is to be independent of the given database and easily portable to other DBMSs.

MADIS supports the pluggability of protocols, i.e., a suitable protocol (e.g., with eager or lazy update propagation, optimistic or pessimistic concurrency control, etc) can be freely chosen, plugged in and exchanged, according to the shift-

*Supported by the Spanish grant TIC2003-09420-C02 and the Generalitat Valenciana grant Grupos05/067.

ing needs of given applications, even at runtime. Protocol switching is seamless and fast, since the meta data for each protocol in the WADIS repertoire is readily at hand at plug-in time.

In WADIS, the more general MADIS protocol repertoire is trimmed to internet-based wide area networks. Globally distributed repositories are a typical case. Replication consistency characteristics of global repositories relying on WANs usually are quite different from ethernet-based or wireless LANs. Characteristics of different kinds of protocols are surveyed in [7].

Section 2 describes the WADIS architecture in general. Section 3 describes the schema extensions effected by WADIS. Section 4 outlines an implementation of the CM as a standard JDBC driver. Section 5 compares our approach with others. Section 6 concludes the paper.

2 Architecture

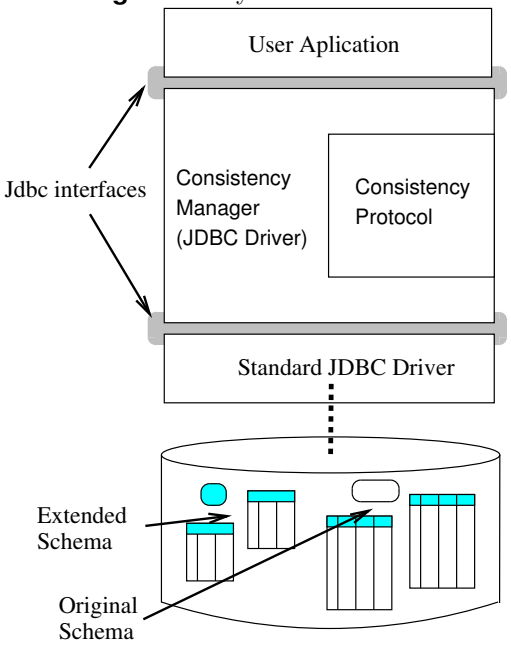
The lower layer of the WADIS architecture caters for schema modifications of the underlying database, involving the creation of additional tables called *report tables*. The upper layer handles transaction requests from users or applications and uses the report tables for transparent replication management.

The report tables account for transactions in the local node, and are updated within the transactions accounted for. The schema extension also includes some stored procedures which hide some schema extension details to the upper layer.

The upper layer is sandwiched between client applications and database, acting as a database 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 tables 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 implementation of the CM, i.e., the core of WADIS, is independent of the underlying database. We are going to describe a Java implementation, to be used by client applications as a common

Figure 1. Layered WADIS architecture.



JDBC driver. Its consistency control functionality is provided transparently to users and applications. The CM handles transaction requests, including multiple sequential transactions in different JDBC consistency modes, and communicates with database replicas. It provides the plug-in for the replication protocol chosen according to given needs and requirements. All supported protocols share some common characteristics. Communication between the database replica is controlled by the CM which is local to each network node.

3 Schema Modification

The lower WADIS layer consists of an extension of the underlying database schema. Distribution of a given database is initiated by an automatic migration of the schema to each replicated node. Migration includes the creation of tables, views, triggers and database procedures for maintaining records about activities performed at transaction time. In particular, writesets and other transaction meta data are recorded. As different meta data are needed by different protocols, the extension caters for the meta data of each protocol in the WADIS repertoire, also of those that are currently not plugged in. Optionally, also information

about readsets (possibly including the information read to perform queries) can be maintained. If that option is not taken, any protocol that would need such information has to perform some work that otherwise is done on the upper layer.

3.1 Modified and Added Tables

To each table T_i in the original schema, WADIS adds a field *local_T_i_id* for identifying and linking each row of T_i with its associated meta data. This identifier is local to each node, i.e., each row may have different *local_T_i_id*'s distributed over the network. Each row also has a unique global identifier, composed of the row's creator node ID and the row identifier local to that node, which is equal in all replicas.

For each T_i, a table `WADIS_Meta_`T_i is created, containing the meta data for any replication protocol in the repertoire. `WADIS_Meta_`T_i contains

- **local_id** local identifier; primary key.
- **global_id** unique global identifier.
- **version** the row's version number.
- **transaction_id** ID of the last transaction that updated the row.
- **timestamp** most recent date the row was locally updated.

In general, it contains all the information needed by any protocol in the WADIS repertoire. Hence, as all fields are maintained by the database manager, any such protocol is suitable to be plugged in deliberately.

In addition to meta tables, WADIS defines a table *WADIS_TrReport* containing a log of all transactions, with the following attributes:

- **tr_id** transaction identifier; part of primary key.
- **global_id** global row identifier; part of primary key.
- **field_id** Optional accessed field identifier; part of primary key.
- **mode**. access mode (read/insert/delete/ modified).

For each transaction, one record for each field of each row is maintained in the `MADIS_TrReport` table. Once a transaction τ is committed, the consistency manager eliminates any information related to τ from `MADIS_TrReport`. Note that several MVCC-based DBMSs do not use locks with row granularity, but block access to entire pages or even tables. Such systems must use multiple "per transaction" temporary *TrReport* tables.

3.2 Triggers

WADIS introduces a set of trigger definitions in the schema. they can be classified in three groups:

- *Writeset managers* for collecting information related to rows written by transactions.
- *Readset managers* for collecting information related to rows read by transactions; their inclusion is optional.
- *Metadata automation* for updating meta data in meta tables.

The writeset collection uses, for each table T_i in the original schema, triggers which insert information related to write-accesses to T_i at transaction time in the `TrReport` table.

The following example shows such a trigger, for the insertion into `mytab` (say). With `getTr_id()`, it gets the transaction's identifier. A row is inserted in the `TrReport` table for each insertion to `mytable`, in order to keep track of the transaction. Deletions and updates are handled analogously.

```
CREATE TRIGGER WSC_insert_mytab
BEFORE INSERT ON mytab
FOR EACH ROW EXECUTE PROCEDURE
tr_insert(mytab,getTrid(),NEW.l_mytab_id)
```

Collecting readsets is optional, due to high costs and also because some protocols can do without readsets Costs are high since the implementation must laboriously compensate for a lack of `TRIGGER ...BEFORE SELECT` in the SQL-99 standard.

Another group of WADIS triggers is responsible for the meta data management. Whenever a row is inserted, such a trigger also inserts the row in the meta data table *WADIS_Meta_T_i*. Since the creator node's identifier (i.e. the node where the

row was created), can be inferred from the row's `global_id` (i.e. the node where the row was created), and the local identifier in the creator node (maintained in another WADIS meta table), all fields in $WADIS\_Meta\_T_i$ can be filled without intervention of any consistency protocol.

Whenever a row is accessed in write mode, another WADIS trigger updates the meta data of that row in the corresponding meta table, i.e., it updates the `version`, the `transaction identifier`, and `timestamp` of the record in the given meta data table. Conversely, whenever a row is deleted, the corresponding meta data row also is deleted, by yet another WADIS trigger.

In summary, WADIS adds, for each table, three triggers, of type BEFORE INSERT, BEFORE UPDATE, BEFORE DELETE, which cater for transaction report management and meta data maintenance. Optionally, for readset management, the `INSTEAD OF` trigger construct must be used, for redirecting write accesses to approprioate tables.

4 Consistency Manager (CM)

WADIS makes use of ready-made database constructs for consistency management. Automatically generated database triggers collect information about accesses at transaction time for the CM, which in turn is independent of the DBMS. Thus, the CM can be ported from one platform to another with minimal effort. This section sketches a Java implementation of CM and how it makes use of the WADIS schema modification.

In our prototype, a JDBC driver encapsulates an existing PostgreSQL driver, for intercepting user application requests. They are augmented to new request for taking care of the meta data associated to the arguments of the requests. Meta data handling is completely hidden from users and applications. The plugged-in protocol is notified about any application request to the database, including query execution, row recovery, transaction termination (i.e. commit/rollback), etc. Thus, the protocol can easily accomplish its tasks regarding replication consistency.

WADIS intercepts queries by means of encapsulating the `Statement` class. Responding to `createStatement` or `prepareStatement` calls, WADIS generates statements that take care of query execution. For each user application query request, WADIS calls the `processStatement()` operation of the currently plugged-in protocol. The latter updates the transaction report, and may modify the statement by adding the patches needed to retrieve some meta data. However, such modifications are only needed by a few consistency protocols, since the meta data can be retrieved from the report tables, once the original query has been completed. Optimistic consistency protocols do not need such meta data before the transaction has requested a commit. The WADIS query execution process is shown in figure 2.

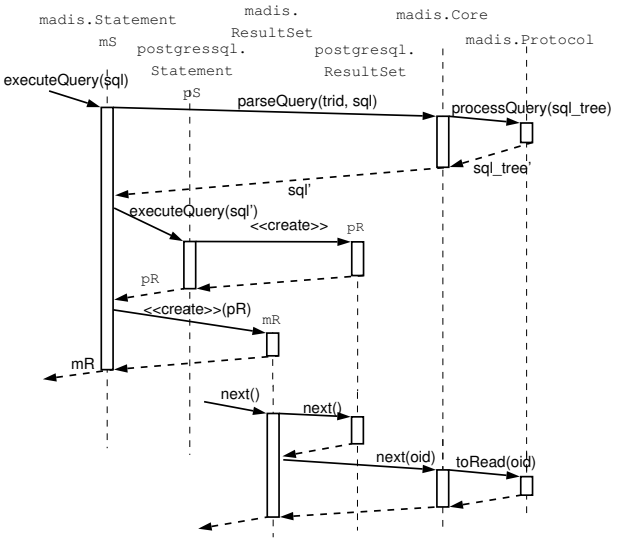


Figure 2. `Query Execution`

Whenever, for a transaction, the user or application requests a *commit* or when a *rollback* is invoked, WADIS notifies the protocol, which thus has the opportunity to involve any replica nodes for satisfying the request. If the protocol concludes its activity with a positive result, then the transaction is ready to commit in the local database, and the CM is notified accordingly, who in turn responds to the user application. Any negative result obtained from the protocol will be notified directly to the application, after the abortion of the local transaction. Similarly, `rollback()` requests are also intercepted, redirected to the CM and forwarded to the protocol.

5 Related Work

With regard to meta data, replication approaches can be classified as *middleware-based* (all work is done by a database-external middleware), *trigger-based* (meta data are collected by triggers and calls to external procedures), *shadow-table-based* (using shadow copies to build update messages for replicas), and *control-table-based* (timestamping each row). Benefits and drawbacks of each are discussed in [9, 10]

In Postgres-R and Dragon [5], a DBMS core is modified to support distribution. This approach strongly depends on the underlying DBMS thus not portable, and must be reviewed for each new DBMS release. However, its performance is generally better than a middleware architecture. In Globdata [3], a middleware providing a standard Java API applications was used. Although protocols are pluggable, the system's API is proprietary, thus impairing the development of applications. Moreover, Globdata's object-orientation turned out to be a drawback. Solutions based on Java, implemented as JDBC drivers, can be found in C-JDBC [8] and RJDBC [1]. The former emphasizes load balancing, the latter reliability. Portability is highly complex. PeerDirect [9] uses triggers and procedures for replication, but no other than a predefined protocol is usable.

6 Conclusion

Different applications require different kinds of replication management. Hence, an adequate choice of appropriate protocols is due. Hence, a middleware which provides flexible support for choosing, plugging in, operating and exchanging suitable protocols is desirable for many applications. This innovative kind of pluggability is being realized in MADIS. As an instance of the latter which is trimmed to the purposes of wide area replication in globally distributed repositories, we have featured WADIS. It disposes of an ample repertoire of replication protocols, each with particular consistency guarantees, from which WADIS allows to plug in, run and exchange suitably chosen ones on the fly. Our implementation makes use of standard SQL-99 constructs such as tables, views, triggers, constraints and stored procedures. Experimental results of a prototype implementation will appear in a follow-up publication.

References

[1] J. Esparza, F. Muñoz, L. Irún, J. Bernabéu: RJDBC, a simple database replication engine. *Proc. 6th ICEIS*, 587-590, 2004.

[2] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. *Proc. ACM SIGMOD*, 173-182, 1996.

[3] L. Irùn, F. Muñoz, H. Decker, J. Bernabéu: COPLA: A Platform for Eager and Lazy Replication in Networked Databases. *Proc. 5th ICEIS*, Vol. 1, 273-278, 2003.

[4] B. Kemme: *Database Replication for Clusters of Workstations*. PhD thesis, ETH Zurich, 2000.

[5] B. Kemme, G. Alonso: A Suite of Database Replication Protocols based on Group Communication Primitives. *Proc. Distributed Computing Systems*, 156-163, 1988.

[6] http://www.iti.upv.es/madis

[7] F. Muñoz, L. Irún, H. Decker: An Overview of Different Approaches to Database Replication. *Encyclopedia of Database Technologies and Applications*, Idea Group, 2005.

[8] http://c-jdbc.objectweb.org

[9] Overview & Comparison of Data Replication Architectures. *Peer Direct* whitepaper, Nov. 2002.

[10] Replication Strategies: Data Migration, Distribution and Synchronization. *Sybase* whitepaper, Nov. 2003.