Asynchronous Replication for Evolutionary Database Development: A Design for the Experimental Assessment of a Novel Approach

Helves Humberto Domingues, Fabio Kon, and João Eduardo Ferreira

Department of Computer Science
University of São Paulo, Brazil
{helves,fabio.kon,jef}@ime.usp.br
http://www.ime.usp.br

Abstract. Environments with frequent changes in application requirements demand an evolutionary approach for database modeling. The challenge is greater when the database must support multiple applications simultaneously. An existing solution for database evolution is refactoring with a transition period. During this period, both the old and the new database schemas coexist and data is replicated in a synchronous process. This solution brings several difficulties, such as interference with the operation of applications. To minimize these difficulties, in this paper we present an asynchronous approach to keep these schemas updated. This paper presents the design for an experimental assessment of this novel approach for evolutionary database development.

Keywords: database evolution, asynchronous data replication, agile methods, performance evaluation

1 Introduction

In the conventional approach for database modeling [9], the complete conceptual, logical, and physical models must be developed before starting to write the application code. This method is often not effective to handle the complexity of many application domains and the speed of changing business requirements. An alternative is to use evolutionary modeling [1]: an iterative and incremental process to create the business data model. However, this modeling process generates the need to work with evolutionary databases [2,10,13] and each data modeling iteration has to deal with important production data, which must be preserved. The evolutionary database must take into account this legacy and allow controlled and organized changes with the data model.

There are studies that deal with schema evolution [4], not worrying about existing data in the database. There are others concerned with the evolution of applications that use this database [11]. When the database is object-oriented, we can find an extensive literature [15,14] on this subject. However, studies about evolutionary relational databases are rare. An exception is Ambler's book, *Refactoring Databases: Evolutionary Database Design* [3], which addresses evolutionary relational databases using refactorings - small changes to improve the internal structure without adding new functionality. Database refactorings have a transition period in which the old and new schemas coexist, allowing the updated applications and those still on maintenance, to work on the same database simultaneuously. To keep the data of the two schemas (new and old) it is necessary to develop a support code. This code, as proposed by Ambler, is a synchronous data replication process using triggers [9]. However, this synchronous process of updating schemas presents several difficulties. First, the programmer must write a specific trigger code for each refactoring; second, avoid ciclic sequences of triggers; third, deal with the increase in the transactions response time due to additional data replication code; and, finally, perform error handling.

Because of difficulties in writing the support code in the form of synchronous triggers, an alternative is to organize and structure the support code in three steps: collection, mapping, and execution. The first step is performed by triggers, which are used only to collect all transaction information. The data replication between the schema is performed by the execution step, following the mapping defined in the second step and using the data collected in the first. These three stages - collection, mapping, and execution - do all the work expected by the support code. We call synchronous replication the case in which these three steps are in the triggers, as proposed by Ambler. When only the data collection is performed with the application transaction and the other steps are performed later, we call it asynchronous replication. This new organization in three steps allows the development of refactoring tools and thus decreases the difficulties in evolving databases. This approach was presented by the authors, in Portuguese, in the Brazilian Symposium on Databases [8].

The main objective of the current paper is to propose the design of experiments to compare the current solution, synchronous replication, with the solution we proposed, asynchronous replication. We present the system description, the experiment hypothesis, the lab environment, and experimental scenarios.

Organization. First, Section 2 describes related works concerning refactoring and asynchronous replication. Second, Section 3 describes the transition period with the support code and, briefly, asynchronous replication on evolutionary databases. We then present the organization of the experimental performance evaluation (Section 4). Finally, we present our future works (Section 5).

2 Related Work

In this section, we discuss related work on refactoring and asynchronous replication that are necessary to understand our asynchronous replication approach.

Wiesmann et al. conceptually define five stages for the data replication protocol [18]:

1. **Request:** the client submits an operation to one or more replicas.
2. **Coordination of servers:** the servers, responsible for the replicas, communicate themselves to synchronize the transaction execution (ordering concurrent operations).

3. **Implementation:** the client-initiated operation is performed on the replica.
4. **Response:** the operation result is transmitted back to the client.
5. **Run result:** the servers communicate with each others to know the operation result.

According to Wiesmann et al., the protocols for data replication are different in the organization of these five phases. The asynchronous replication is similar to the delayed replication protocol, defined by Wiesmann et al. The advantage of this replication model is the low response time to customers, but the disadvantage is an eventual reconciliation process that must be executed to resolve conflicts. Comparing the asynchronous replication with this approach, we have three different points. First, the nomenclature: we use the term asynchronous model, not the *deferred* term. Second, we add the mapping phase to make the necessary changes in the transaction before updating the replica. Finally, we consider that the implementation phase is when the replicas are updated and the reconciliation task, if necessary, is executed.

In recent works, we can find frameworks to assist the schema evolution in temporal databases [6]. Curino et al. propose a framework named *Panta Rhei* designed to (i) provide database administration tools to facilitate the schema development, (ii) enable automatic old query rewrite to work in newer schemas, (iii) allow efficient historical data and metadata archiving, and (iv) allow complex temporal queries on such historical information. This framework is based on the work on the PRISM system [12] that defines the schema change operators. The asynchronous replication database has the same motivation reasons that Curino et al. have. Information systems with many and frequent changes in the database model need development tools to manage these changes. However, while the proposal by Curino et al. is based on automatic query rewrite to allow access to old schemas, our proposal uses asynchronous data replication during the transition period.

3 The Asynchronous Replication Approach

Due to limitations of the support code implementation in the form of synchronous triggers, an alternative is to reorganize the support code in three steps: collection, mapping, and execution. In this section, we briefly describe this approach that was presented by the authors in SBBD [8].

The **transition period**, when the old and new schemas coexist in the database, requires a support code to keep the two schemas synchronized. In the simplest case – when data changes occur only in the old schema, and the new one is used only for queries – we have a support code that must replicate all the changes from the old model to the new one. Thus, the code support allows applications not to be affected by the existence of two schemas – they always access updated data. At the end of the transition period, when all the applications that access the database have adapted to the new schema, the old schema and support code are destroyed. Because it is a code that will be used for a short period, this code is expected to be easy, fast, and simple to write.

The **collection step** is performed by a simple and generic trigger, called collector trigger. This trigger is used for all types of existing refactorings, with the goal of capturing all the transaction information. Captured information are the operation type (insert, delete, or update) and the previous and new values in the table. We do not use the database log to collect the transaction information because we want to use any database management system that has triggers. Database log files have different formats for each database management system.

The purpose of the **mapping step** is to define a mapping from the source to the target table. Each map represents one database refactoring that needs data replication.

In the **execution step**, the replication process is executed and the old and the new schemas are updated. The goal is to consume the information collected in the first step, following the mappings defined in the second one.

Figure 1 shows all asynchronous replication actions. The collector trigger writes the transaction information from the source table (1). The replication process reads the map (2), consumes the transaction information (3), and updates the target table (4).

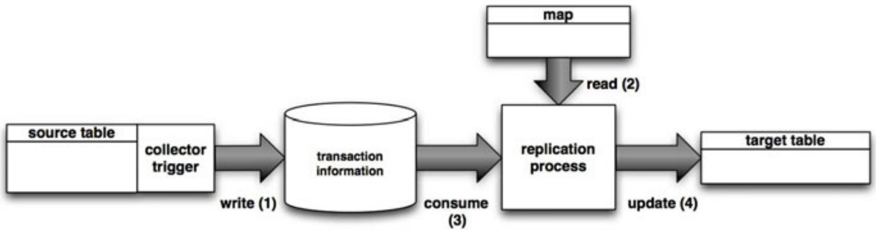

Fig. 1. Asynchronous replication approach

4 Proposed Experimental Assessment

To analyze the performance of the asynchronous alternative, we organized an experiment to compare it with Ambler's solution: the synchronous data replication for refactoring [3]. The comparison must be carried out by checking, at various levels of database concurrency, the performance of both solutions. Because asynchronous data replication is delayed, the time required to process a pending update must be measured to complete the assessment of the proposed approach.

According to Wohlin et al. [19], the technique that is appropriate for comparing the synchronous and asynchronous methods is the *engineering experiment.* This kind of experiment is defined as the method that observes the solutions, suggests the most appropriate solutions, develops, measures, analyzes, and repeats this process until no further improvement is possible.

4.1 System Description and Experiment Hypotheses

We designed the simplest and most representative environment to simulate one database refactoring for performing the comparison between the two approaches. For that, we used only one database table (`meetings`), which was the most complex one found in a specific system for the healthcare domain [7] developed by the authors. This table is the center of care modeling provided in public health centers and contains 19 attributes to store the necessary information.

The aim of the experiment is to validate the following hypotheses:

- **Hypothesis 1:** the synchronous method generates many locks and the amount is significantly greater than the one in the asynchronous method.
- **Hypothesis 2:** a system that updates the database with asynchronous replication has a better performance, measured in number of operations per second, than when using the synchronous method.
- **Hypothesis 3:** The time to process a pending operation in the asynchronous method is small and is on the order of tens of milliseconds.

Hypotheses 1 and 2 focuses on the comparison between synchronous and asynchronous methods in terms of performance. The first one aims at analyzing the amount of locks while the second considers the number of operations per second. Hypothesis 3 deals with the period of inconsistency in the database table. There will be no more inconsistency only after the execution of the replication process. The average processing time of a pending operation will provide an estimate of this inconsistency period.

4.2 The Laboratory Environment

In this section, we describe the virtual machine and the real machine created to be the testbed environment that is as close as possible to a real environment. The main elements of this laboratory environment are shown in Figure 2. The use of a virtual machine in the lab environment is very useful since we can easily manage all the resources of this machine. Below, we describe all parameters of our lab environment.

Virtual machine - The virtual machine was created with the software *VMware Fusion* [17], version 3.1.2, and has the following elements:

1. **Resources:** contains 1GB memory, 500GB hard disk with one of two processor cores of Core 2 Duo 2.8Ghz.
2. **Operating system:** uses GNU/Linux distribution, Ubuntu 9.10 Server 64-bit standard installation.
3. **Database manager:** PostgreSQL version 8.4, with a standard installation. The only changed parameter was the maximum amount of connections. We set the value to 200, due to the amount of virtual machine memory.

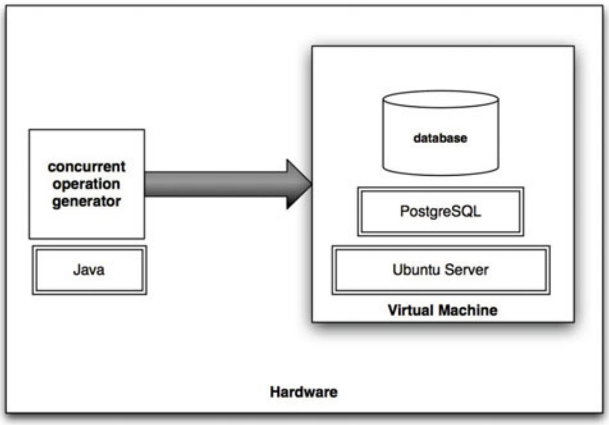

Fig. 2. The lab environment

Real machine - The hardware is a MacBook Pro with the following elements:

1. **Hardware:** contains 7GB of memory, 20GB hard disk with Core 2 Duo 2.8Ghz.
2. **Operating system:** Mac OS X version 10.6.4 64 bit.
3. **Concurrent operation generator:** the *Bristlecone Performance Test* [5] tool was used to generate a large volume of concurrent operations in the database.
4. **Java Virtual Machine:** the JVM used is the *SE Runtime Environment 1.6.0.22* [16]. Basically, the JVM was used to run the *Bristlecone Performance Test* tool. All client processes that generate load on the environment are actually Java threads.

Throughout the experiment, both the real machine and the virtual machine will be monitored by operating system tools (*Activity Monitor* for Mac OS X and *top* for GNU/Linux) to check for a lack of resources that could compromise the experiment.

4.3 Scenarios

In this section, we present three experiment scenarios. All scenarios have the same concurrent operation generator process. What differentiates each scenario is the existence of triggers and their type: synchronous or asynchronous.

We set up two tables (`meetings_0` and `meetings_1`) with the same structure but with different names and will perform data replication between them. The `meetings_0` table will be the source and `meetings_1` table, the target.

Below, we have the Figure 3 and the description of each scenario.

– **Scenario 1 - no trigger**: the two meetings tables do not have data replication, ie, there is no trigger associated with the two tables. This scenario aims to be the baseline.

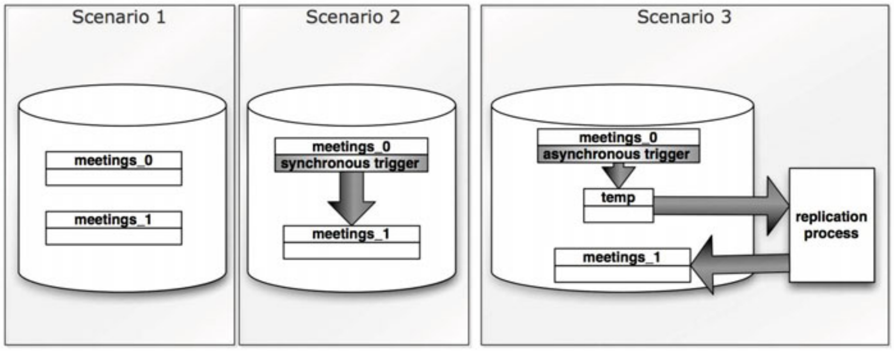

Fig. 3. Scenarios

– **Scenario 2 - synchronous trigger**: The purpose of this scenario is to evaluate the performance of the current solution for database refactorings. This trigger is as simple and efficient as possible: it only performs the update to `meetings_1` table from the `meetings_0` table.
– **Scenario 3 - asynchronous trigger**: The purpose of this scenario is to evaluate the asynchronous replication method. The asynchronous trigger records data operations into a temporary table (`temp`) and the replication process runs simultaneously with the concurrent operation generator process.

5 Future Works

Right after the first entry into database production, the user starts storing and updating the data of its business. Then, for each change in the database, it is necessary to address the existing data. The challenge to evolve one database is even greater when the database is used by multiple applications simultaneously. Each application will have different problems and will spend different times to make the necessary adjustments. A transition period with the old and new coexisting schemas is very useful, but requires a support code to maintain the schemas updated.

The main objective of this paper was to organize an experimental performance comparison between the current solution, synchronous replication, and the solution we proposed, asynchronous replication. We described the experiment objectives, the lab environment, and the scenarios that define the structure of the experiment. In a future paper we will describe the experiment execution and present its results, the data analysis, and the interpretation of experimental results.

References

1. Ambler, S.W.: Agile modeling: effective practices for eXtreme programming and the unified process. John Wiley and Sons (2002)
2. Ambler, S.W.: Agile Database Techniques. John Wiley & Sons Inc. (2003)

3. Ambler, S.W., Sadalage, P.J.: Refactoring Databases: Evolutionary Database Design (Addison Wesley Signature Series). Addison-Wesley Professional (2006)
4. Balsters, H., de Brock, B., Conrad, S. (eds.): Database Schema Evolution and Meta-Modeling. Springer, Heidelberg (2001)
5. Continuent: Bristlecone performance test, `http://www.continuent.com/community/lab-projects/bristlecone` (last access on February 18, 2011)
6. Curino, C., Moon, H.J., Zaniolo, C.: Managing the history of metadata in support for db archiving and schema evolution. In: Fifth International Workshop on Evolution and Change in Data Management, pp. 78–88. Springer, Heidelberg (2008)
7. Domingues, H., Correia, R., Kon, F., Kon, R., Ferreira, J.E.: Análise e modelagem conceitual de um sistema de prontuário eletrônico para centros de saúde. In: SBC - Workshop de Informática Médica, pp. 31–40. Belém, Brasil (2008)
8. Domingues, H., Kon, F., Ferreira, J.E.: Replicação assíncrona em modelagem evolutiva de banco de dados. In: SBBD - XXIV Simpósio Brasileiro de Banco de Dados (Brazilian Symposium on Databases), pp. 121–135. Ceará, Brazil (2009)
9. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, quinta edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2006)
10. Fowler, M.: Evolutionary database design (2003), `http://www.martinfowler.com/articles/evodb.html` (last access on February 18, 2011)
11. Hick, J.-M., Hainaut, J.-L.: Database application evolution: A transformational approach. Data & Knowledge Engineering 59(3), 534–558 (2006)
12. Moon, H.J., Curino, C.A., Deutsch, A., Hou, C.-Y., Zaniolo, C.: Managing and querying transaction-time databases under schema evolution. Proceedings of the VLDB Endowment 1(1), 882–895 (2008)
13. Oertly, F., Schiller, G.: Evolutionary database design. In: Fifth International Conference on Data Engineering, pp. 618–624 (1989)
14. Rahm, E., Bernstein, P.A.: An online bibliography on schema evolution. SIGMOD Rec. 35, 30–31 (2006), `http://doi.acm.org/10.1145/1228268.1228273`
15. Rashid, A., Sawyer, P.: A database evolution taxonomy for object-oriented databases. Journal of Software Maintenance and Evolution: Research and Practice 17(2), 93–141 (2005)
16. Support, A.: Java for mac os x 10.5 update 8. `http://support.apple.com/kb/DL971` (last access on February 18, 2011)
17. VMware: Desktop products, vmware fusion 3, `http://www.vmware.com/products/fusion` (last access on February 18, 2011)
18. Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G.: Understanding replication in databases and distributed systems. In: Proceedings of ICDCS 2000, pp. 264–274. IEEE Computer Society (2000)
19. Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslen, A.: Experimentation in Software Engineering: An Introduction. Kluwer (2000)