The ZooLib Tuplebase: An Open-Source, Scalable Database Architecture for Learning Sciences Research

Chris Teplovs, Institute for Knowledge Innovation and Technology, OISE/UT,
chris.teplovs@utoronto.ca
Andrew Green, The Electric Magic Company, ag@em.net
Marlene Scardamalia, Institute for Knowledge Innovation and Technology, OISE/UT,
mscardamalia@kf.oise.utoronto.ca

Abstract: We introduce a free, open-source, non-relational database architecture that is robust, mature, scalable and extensible. The absence of a fixed schema is particularly important to research in the learning sciences where a broad spectrum of research data may need to be stored. We highlight an alternative query mechanism that is suited to highly interactive applications.

Introduction

The learning sciences are characterized by perspectives from multiple disciplines and the concomitant diversity of perspectives on research and the range of data collected and stored. The availability of database management systems (DBMSs) is equally diverse. Popular DBMSs include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server, all of which are relational; that is, they are based on tables. Our experience with database-driven environments has led us to conclude that relational databases have a number of constraints that make them less than ideal for learning sciences research. Perhaps most importantly, the relational nature of these databases means that a predetermined schema must be in place. Of course, this schema can be changed, but that often leads to versioning problems with a released, installed program base. As research needs evolve, users need a flexible environment with no predetermined schema.

The ZooLib Tuplebase

Our solution was to utilize a tuplebase. The ZooLib Tuplebase (Green, 2002) is derived from the tuplespace concept initially explored in the Linda coordination language (Gelernter and Carriero, 1992), another well known derivation of which is Sun's JavaSpaces (Marmoud, 2005). Since 2002 it has provided the infrastructure for Knowledge Forum, the original Knowledge Building Environment (Scardamalia and Bereiter, 2003). Whereas JavaSpaces is Java-only and relies on many of that language's features, ZooLib's tuplebase works with C++ and Java, and is well suited to work with other languages. A tuple, for the purposes of this paper, is simply a collection of zero or more property-value pairs. Values are typed and can include strings, numbers, vectors, dates, or raw data. Full support for international character sets is built into the system. A sample tuple might look like:

```
{ Object = "author"; fnam = "John"; lnam = "Doe"; unam = "jdoe"; pass = "xxx"; }
```

A tuplebase is an array of 2^{64} tuples. It's always fully populated so that every index of the array contains a tuple. When first created a tuplebase contains 2^{64} empty tuples, tuples with no properties. The 64 bit index of each tuple is unique and permanent and is more usually called its ID. Individual tuples are accessed by ID, and more usefully sets of tuples can be accessed by using iterators that return tuples matching criteria.

All accesses to a tuplebase, even simple reads, are made in the context of a transaction. When the transaction is committed failure can be reported. The power of this is that code can treat the tuplebase as always being in a consistent state without any need to explicitly synchronize access. The same applies when the transaction needs to write to the tuplebase -- one can write code of arbitrary complexity, and when committed either all of the work is made permanent and visible to the world or none of it is. The underlying storage is currently implemented as a RAM-based layer that is periodically checkpointed to disk, or as an Oracle Berkeley DB.

The tuplebase is queried directly through Java or C++. A comparison of simple SQL and ZTB queries is shown in Table 1.

Table 1: Comparison of SQL and ZTB queries in Java.

SQL	ZTB

ResultSet rs = stmt.executeQuery("SELECT * FROM author WHERE lnam= 'doe'");
while (rs.next()) {
String s = rs.getString("firstName");
}
	ZTBSpec s = ZTBSpec.sEquals("lnam","doe");
	for (ZTBIter it = new ZTBIter(txn, tb, s);
	it.hasValue(); /* no inc */) {
	String s = it.get().getString("firstName");
	}

The ZooLib TupleStore Watcher

Traditionally, databases provide a query mechanism that returns a result set based on a query specification. SQL is the most common database query language. Our tuplebase implementation augments this model by providing the ZooLib TupleStore Watcher (ZTSWatcher). ZTSWatcher lets an application register its interest in the result sets of any number of queries, and keeps the application informed as those result sets change. This information can be used to trigger normal transactional work. ZTSWatcher is also used as the basis of a hierarchical scaling architecture, and of the 'Soup' API that cleanly integrates atomic tuplebase operations with traditional event-driven user interfaces.

Case Study: Layering research data into a live system

We regularly conduct analyses of writing contributed by participants in Knowledge Forum. In the past, these data were stored in an adjunct database, rather than within the system being researched. Whereas that is common practice there are some notable shortcomings. For example, data in the live system can change and invalidate the companion research data. Consider the case of simple writing metrics (e.g. word count). When participants update their notes, the research data may become invalidated. In our research systems we utilize a ZTSWatcher that is set up to detect cases where we don't have up-to-date analyses and immediately spawn a process to update the data. Typical calculations range from simple text processing to advanced use of latent semantic analysis. The power of the tuplebase approach is that data that is irrelevant to a particular application that is accessing the tuplebase is simply ignored. Thus we can add a numerical representation consisting of several hundred floating-point numbers to a message body without adversely affecting the system. Similarly, researchers can annotate knowledge objects without having any visible effect on the participants' discourse space. We are starting to experiment with using the tuplebase to share analytic results. Using this approach researchers can use the results of one analysis to feed into others.

Comparison to other database technologies

Our experience leads us toward concurring with the view that the "relational databases for all" paradigm may be coming to an end (Stonebreaker et al., 2007). Researchers are finding that specialized databases tuned to specific domains (e.g. data warehousing, text management, and scientific databases) consistently outperform their relational brethren. A variety of alternative database engines are becoming available: CouchDB (http://couchdb.org) is an example of a new non-relational database management system. The ZooLib Tuplebase, by contrast, has been in deployment for over 5 years and is a stable, mature technology. It is available as free, open-source software via http://zoolib.org/.

Why not just use a Resource Description Format (RDF) store? RDF is a specification that underpins Semantic Web technologies (W3C, 2004). Whereas it is powerful and extensible, it is difficult to implement a manageable storage layer in a straightforward manner. RDF stores consist of generic subject-predicate-object triples which, although necessary for Semantic Web applications, impose a level of complexity that is unnecessary for most data storage requirements.

The ZooLib Tuplebase represents an important evolutionary step in database technology and is well-suited to learning sciences research, particularly as we see a move to open-source, interoperable architectures and the sharing of analytic results.

References

Gelernter, D., and Carriero, N. (1992). Coordination Languages and their Significance. Communications of the ACM 35(2): 97-107.

Green,. A. (2002). Tuplebase. http://www.em.net/portfolio/2002/10/tuplebase.html.

Marmood, Q. H. (2005). Getting Started With JavaSpaces Technology: Beyond Conventional Distributed Programming Paradigms. http://java.sun.com/developer/technicalArticles/tools/ JavaSpaces/

Scardamalia, M. and Bereiter, C, (2003). Knowledge building environments: Extending the limits of the possible in education and knowledge work. In A. DiStefano, K.E. Rudestam, and R. Silverman (Eds.), Encyclopedia of distributed learning, Thousand Oaks, CA: Sage Publications.

Stonebreaker, M., Madden, S., Abadi, D.J., Harizopoulos, S., Hachem, N., and Hellard, P. (2007). The End of an Architectural Era (It's Time for a Complete Rewrite). VLDB'07.

W3C. (2004). RDF Primer. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/