Diversity with Off-The-Shelf Components A Study with SQL Database Servers

Peter Popov, Lorenzo Strigini
Centre for Software Reliability, City University, London
{ptp,strigini}@csr.city.ac.uk

Abstract

Fault tolerance is often the only feasible remedy available to a user or integrator when using insufficiently dependable off-the-shelf software products. In particular, modular redundancy with diversity, as e.g. in N-version software, may be an affordable solution, but there has been little study of its practical effectiveness and implementation difficulties with off-the-shelf components. We have started an experiment to help to remedy this situation. We report preliminary observations from the development and early use of the experimental set-up.

1. Background

When confronted with the dependability limitations of off-the-shelf (OTS) software, the best and often the only practical option available to a system integrator is software fault tolerance [1], using modular redundancy with diversity (e.g. N-version software) or checker compo - nents (implemented for instance in the form of wrappers). The alternatives - extra V&V activities or changes to the code - are impossible for most proprietary software, as the source code and design documentation are not available, and too expensive, in most other cases.

The use of wrappers is a reasonably popular solution. Modular redundancy with diverse OTS software is less so, although there is research interest (e.g., [2, 3]), especially for improving security (intrusion tolerance). Its limited application may be due to cost issues (cost of the OTS products, and even more of the possibly difficult integration of complex OTS parts into a redundant configuration) and/or to the difficulty of assessing the effectiveness of multiple version software [4].

To explore these issues and provide evidence for or against diverse modular redundancy for a reasonably complex category of widely-used OTS software, we are running an experiment with SQL database servers. We use a test harness (developed in co-operation with the Technical University in Plovdiv, Bulgaria) to run multiple OTS SQL servers in parallel on the same sequence of inputs (queries) and log any failures. The architecture is schematically shown in Fig. 1.

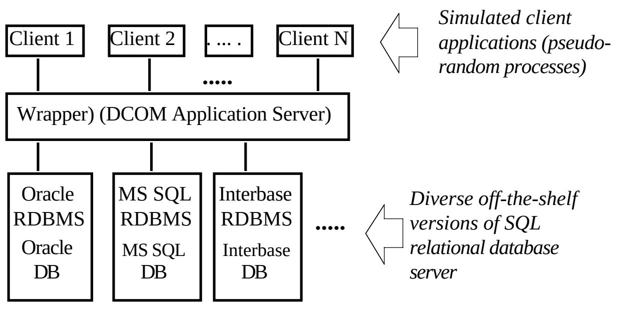
Client 1
Client 2
. ... .
Client N
.....
Simulated client applications (pseudo-random processes)
Wrapper) (DCOM Application Server)
Oracle RDBMS
Oracle DB
MS SQL RDBMS
MS SQL DB
Interbase RDBMS
Interbase DB
.....
Diverse off-the-shelf versions of SQL relational database server

Fig. 1 Architecture for experiments with diverse database servers (simplified view). The wrapper copies the clients requests (SQL queries and control commands) to the multiple databases, collects and compares their responses, and produces log data for the experiment logs. The database products listed are just examples among those that can be used, in any combination of (identical or diverse) servers. The servers are distributed over multiple computers on a LAN, on similar or diverse operating systems.

2. Diverse-redundant server designs

Our experimental implementation is not a full-fledged fault-tolerant server. For instance, it has no automated mechanism for recovery of a corrupted database version.. However, we had to solve some of the same problems that would affect the design of a proper fault-tolerant server. In what follows, it is important to remember that no change was made to the OTS database servers.

Two suitable redundancy schemes are *N-version programming* (NVP) and *N-version self-checking programming* (NSCP), as they are called in [5]. Either can be applied in our set-up, by appropriate configuration of the wrapper component. An NSCP architecture relies on the individual servers guaranteeing clean failures: the server versions only issue correct outputs, or no output. NSCP configurations can thus exploit diversity both for reliability and for speed: the first response produced for a query can be forwarded to the client. With changes to the wrapper, the scheme can be turned into one with stand-by redundancy to reduce load on the platforms (by logging queries and replaying them to the stand-by versions when necessary), or with load balancing (by submitting read-only transaction only to a subset of the servers). The limit of NSCP is its dependence on the assumption of clean or perfectly detectable failures. An NVP configuration avoids this limitation as it uses voting among the versions outputs to both detect and mask (whenever possible) failures of the servers.

The main design problem is how to preserve both consistency among the multiple databases and atomicity of transactions. If the wrapper simply broadcasts each query to the database server versions, each server version will serve them in an order that is affected both by the variable delays of the communication mechanisms and by the different, and unknown, policies applied by the servers to ensure acceptable throughput while preserving transaction atomicity. So, correct processing by all server versions may produce inconsistencies between the database versions. This is avoided by the wrapper enforcing some degree of synchronization, delaying queries as appropriate. For the NSCP scheme, it is sufficient that write transactions be strictly synchronized on all server versions. For voting, in practice the same order must be imposed on the servers for all transactions, bringing significant synchronization overhead. In either case, the overhead cost must be compared against the dependability improvements achieved, and against the cost of a comparably dependable, but not OTS, database server.

Another practical design problem is that the SQL servers implement different subsets or supersets of the SQL-92 and SQL-99 standards, and also differ in details of the syntax of the query language they accept. Since client programs would normally be written with one of the various server types in mind, the wrapper would need both to check that all queries are acceptable to all server versions, and to translate them into their respective di a-lects . Both functions appear reasonably simple to i m-plement. In our own experimental set-up, we simply use the subset of SQL that is common to all the servers we are testing, and the translation is mostly performed off-line.

3. Measuring the efficacy of diversity

We plan to run the servers for long periods of time with various demand profiles, compare the counts of failures affecting only one server against those affecting two or more, and thus estimate the reliability advantage (if any) given by diversity with redundancy. A plan for statistical measurement requires:

- a choice of usage profile: the statistical process of queries and transactions. We plan to use three databases with various reference loads: a warehouse database derived from a real-life application; a simplified banking application, and an implementation of the TPC-C benchmark;
- an oracle for detecting failures. We are using co m-parison of responses from the servers, time-outs, periodical comparisons of database contents, and, for the banking application, checks on invariants on the database contents, guaranteed by the way the simulated clients generate their queries.

The dependability gains that diversity gives depend, for a given usage profile, on the specific redundant configuration (set of server versions used, NVP or NSCP scheme), and on the probability of query sequences that cause some server versions to fail, but few enough for the failure to be masked. Within limits, the effects of different configurations can be estimated from a single measurement campaign, feeding equivalent sequences of queries (streams of queries, translated into the server versions dialects and synchronized to respect certain constraints) to the multiple OTS servers. We can expect many failures to be due to Heisenbugs [6] , i.e., dependent on rare coincidences between states of the databases, the applications and the operating systems of the server computers. So, reliability may be influenced by details such as the physical allocation of server processes to computers. In particular, we will take care to discriminate any gain due specifically to diversity from the gain due to the simple separation of redundant servers on separate computers.

A preliminary evaluation step concerns *fault* diversity rather than *failure* diversity. By manual selection of test cases, one can check whether the redundant configuration would tolerate the known bugs in the lists of bugs reported for the various OTS servers. This phase is well under way and producing encouraging results, although these cannot be translated into measures of reliability increases. For instance, out of 62 bugs reported for two OTS servers over a period of one year, and affecting a common subset of the SQL language, there were only 4 for which coincident failures of the two servers were produced.

Acknowledgments

This work was supported by the U.K. Engineering and Physical Sciences Research Council within project DOTS, Diversity with Off-The-Shelf Components . The test harness was built by A. Kostov, V. Mollov and D. Selensky. Ilir Gashi performed the fault diversity testing.

References

[1] P. Popov, L. Strigini and A. Romanovsky, "Diversity for off-the-Shelf Components", DSN 2000 Fast Abstracts supplement, New York, NY, USA, 2000, pp. B60-B61.

[2] J. Reynolds, J. Just, et al., "The Design and Implementation of an Intrusion Tolerant System", Proc. DSN 2002, Washington, D.C., USA, 2002, pp. 285-292.

[3] P. Verissimo, N. F. Neves and M. Correia, "The Middleware Architecture of MAFTIA: A Blueprint", Proc. IEEE Third Information Survivability Workshop (ISW-2000), Boston, Massachusetts, USA, 2000.

[4] B. Littlewood, P. Popov and L. Strigini, "Modelling software design diversity - a review", ACM CS, 33, pp. 177 - 208, 2001.

[5] J. C. Laprie, J. Arlat, et al., "Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures", IEEE Computer, 23, pp. 39-51, 1990.

[6] J. Gray, "Why do computers stop and what can be done about it?", Proc. SRDSDS-5, Los Angeles, CA, USA, 1986, pp. 3-12.