Efficient Testing of High Performance Transaction Processing Systems

Dennis Wildfogel
Tandem Computers Inc.
Cupertino, CA 95014
wildfogel_dennis@tandem.com

Ramana Yerneni
Computer Science Dept.
Stanford, CA 94305
yerneni@cs.stanford.edu

Abstract

Testing the reliability of high performance transaction processing systems poses many difficult challenges that are not adequately answered by conventional testing techniques. We discuss a new test paradigm, which is dynamic and exploratory in nature, and discuss its ability to meet these challenges. We describe an implementation of this paradigm in products that aid in efficiently testing reliable, high performance transaction processing systems at Tandem Computers Inc.

1 Introduction

Mission critical database applications, e.g., a stock market system, insist on guaranteed data integrity in all situations. They also demand extremely high performance and high system availability (the cost of even short outages, of the order of minutes, is prohibitively high). Such applications employ high performance transaction processing systems that also have high availability characteristics ([TDG87], [SKPO88], [MHLPS92]). Tandem's NonStop SQL ([TDG87], [TPG88], [BP88]) is one such system. NonStop SQL runs in a shared nothing, multi-processor architecture that provides high performance through scalable parallelism ([Bar81], [Sto91]). High availability is afforded through hardware fault tolerance (by way of redundancy, e.g., mirrored disks) as well as software fault tolerance (by way of process pairs with active, "hot standby" backup processes) ([GR93], [Bor84]). At the heart of the NonStop SQL system, the Transaction Manager (TM) and the Disk Process (DP), are two products that work together to provide ACID — atomic, consistent, isolated, durable – transactions, thus ensuring data integrity at all times.

An analysis of outages of transaction processing systems shows that the dominant cause of system unavailability is software faults (they far outweigh all other faults put together, including operator faults and hardware faults) ([Gray90], [GR93]). Thus, it is extremely important to ensure the reliability of the software components of NonStop SQL, in order for it to be a highly available transaction processing system. TM and DP are two of the key software components whose reliability is of particular concern because faults in these components can compromise database consistency (by violating the ACID semantics of transactions), not just system availability.

Developing software components like TM and DP poses many challenges. One particularly difficult task is to test their reliability. In this paper, we argue that conventional solutions to testing software products are inadequate in verifying the reliability of products like TM and DP. We describe a novel dynamic testing paradigm that is more suitable for this task. We discuss how tests can be written in this paradigm to efficiently and effectively test high performance, highly available transaction processing systems like NonStop SQL.

2 Conventional Approaches

Conventional approaches to testing software products are based on a static paradigm of specifying the precise set of inputs and outputs of each test ([Mye79]). There are three main problems with conventional approaches

Proceedings of the 23rd VLDB Conference
Athens, Greece, 1997

when testing products whose behavior is influenced by the state of the system in which they operate. They are:

- it may not be possible to specify each test completely because part of the input to the test is the system state that influences the operation of the products, and one may not be able to describe the relevant system state precisely in a high level specification of the test. This makes test specification cumbersome and arduous.

- it is very difficult, if not impossible, to create the specified system state accurately when executing a test. This makes test development and execution very difficult.

- the number of system states that may affect the products is so large that it is very inefficient, and often infeasible, to enumerate all the relevant tests. This makes the whole testing process inefficient.

- it is nearly impossible to recognize in advance many different states that have nontrivial affect on the operation of the products. This makes the testing process potentially deficient because it may miss some important test cases.

TM and DP implement many key protocols and algorithms like Two-Phase Commit, Two-Phase Locking, Write Ahead Logging, Do-Undo-Redo Recovery, and Steal-No Force Buffer Management ([MHLPS92], [BN97]). The operation of these protocols and algorithms is influenced by the state of the system at the time of their execution. In this sense, TM and DP are highly state sensitive products.

TM and DP run as process pairs (TM-Primary, TM-Backup, DP-Primary and DP-Backup), that are deployed in different CPUs, in order to tolerate the failure of any single component of the system, without compromising service availability. When the primary process, or the CPU in which it resides, fails the backup process takes over and resumes the product services in a seamless manner. Process checkpointing mechanisms are used to implement the ability to take over no matter when the failure occurs. Of course, the behavior of the take over code will be strongly influenced by the state of the primary and the backup processes at the time of the failure event. This is another aspect of the state sensitivity of TM and DP.

Thus, given the state sensitive nature of TM and DP, conventional test approaches are not suitable to verify the reliability of TM and DP.

3 Dynamic Test Paradigm

We developed a dynamic test paradigm, which is much better suited to the task of verifying the reliability of TM and DP ([Wil92]). The essential idea of this test paradigm for transaction processing systems is to create transaction activity in the system, asynchronously generate system events (like the failure of a hardware component, e.g. a CPU), and verify the appropriate response of the system with respect to data integrity, availability and performance.

We illustrate the dynamic test paradigm with the following example. NonStop SQL provides complete tolerance of single point failures. That is, the system availability is not affected by the failure of any one component of the system. Of course, the database consistency must always be preserved, no matter what failures happen. So, for example, a dynamic test would create background transaction activity in the system, invoke the failure of a single CPU in the cluster (happening asynchronously with respect to the background transaction activity), and verify that all the system services are available when the CPU is up (to start with), when it fails, and when it comes back up. In addition, it would also verify that the consistency of the database is maintained.

We note here that the timing of the CPU failure in the above example test, with respect to the state of the various transactions active in the system and the states of the processes like the TM and the DP at the time of the CPU failure, is crucial. Since the generation of the CPU failure event in the test is asynchronous to the transaction activity created by the test, each run of the test may produce a different timing situation. In order to robustly verify the correct behavior of the NonStop SQL system in the presence of a CPU failure, one may run this test many times over.

Another approach, and the one we have taken in our implementation, is to repeat the failure event generation many times in a single test. That is, in the example test described above, the test will take the CPU down, bring it up, take it down, bring it up, and so on. At various points in the test, the availability of the system services as well as the consistency of the database may be verified. Thus, in the dynamic test paradigm, it is common to have tests that repeat the system events many times, within a test.

4 Implementation Experience

Each time a system event occurs in a dynamic test, one can define a test case based on the timing of this event with respect to the state of the system (the transactions and the processes). In this sense, each dynamic test spawns many test cases in a single run of the test. Note here that each of these test cases is not explicitly specified. Instead, each test covers a large set of test cases that are enumerated dynamically at run time. This exploratory nature of the dynamic test paradigm clearly distinguishes it from conventional approaches that require explicit specification of the test cases. The ability of a dynamic test to cover a large number of test cases is a powerful feature which leads to concise test specification, simple test design, and efficient test development and execution. Our implementation of the dynamic test paradigm in verifying the reliability of TM and DP has demonstrated this fact clearly.

Over the past few years, we have relied heavily on the dynamic test paradigm to test TM and DP ([Wil91]). During this time, the reliability of these products has improved significantly. This, and the fact that tests developed using the dynamic paradigm have found many faults in these products, points strongly to the significant contribution of our paradigm in ensuring the reliable development of TM and DP.

Some aspects of the TM and DP products, like their basic application programming interface and system administration interface (to configure, monitor and administer) are amenable to verification based on conventional testing approaches, based on the static test paradigm. Accordingly, the test libraries developed using the dynamic test paradigm are complemented by sets of tests based on conventional approaches to provide a comprehensive verification of the reliability of these products.

We have employed the dynamic test paradigm in verifying the reliability of other related products like NonStop SQL's Disaster Recovery Facility ([Lyon90]). Our experience suggests that this test paradigm is very suitable for the reliability verification of many products, not only in transaction processing systems but also in contexts like operating systems and network management systems.

5 Conclusion

In this paper, we have briefly described the problem of efficiently testing high performance transaction processing systems. We observed that conventional software testing techniques are not adequate in testing the core software components of such systems. We discussed a novel test paradigm that is dynamic and exploratory in nature. Many years of our experience in its use have shown its applicability in efficiently testing Tandem's NonStop SQL, a high performance, highly available transaction processing system.

References

[Bar81] J. Bartlett. A NonStop Kernel. *Proc. 8th Symposium on Operating Systems*, December, 1981.

[BN97] P. Bernstein, E. Newcomer. Principles of Transaction Processing: For the Systems Professional. Morgan-Kauffman, 1997.

[Bor84] A. Borr. Robustness to Crash in a Distributed Database: A Non Shared-Memory Multi-Processor Approach. *Proc. 10th International Conference on Very Large Data Bases*, September, 1984.

[BP88] A. Borr, F. Putzolu. High Performance SQL Through Low-Level System Integration. *Tandem Computers TR 88.10*, June, 1988.

[GR93] J. Gray, A. Reuter. Transaction Processing: Concepts and Techniques. Morgan-Kauffman, 1993.

[Gray90] J. Gray. A Census of Tandem System Availability between 1985 and 1990. *IEEE Transactions on Reliability*, October 1990.

[Gray81] J. Gray. The Transaction Concept: Virtues and Limitations. *Proc. 7th International Conference on Very Large Data Bases*, June 1981.

[Hel89] P. Helland. The TMF Application Programming Interface: Program to Program Communications, Transactions, and Concurrency in the Tandem NonStop Computer System. *Tandem Computers TR 89.3*, 1989.

[Lyon90] J. Lyon. Tandem's Remote Data Facility. *Proc. 35th IEEE Compcon*, 1990.

[MHLPS92] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, P. Schwartz. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. *ACM Transactions on Database Systems*, 17(1), 1992.

[Mye79] G. Myers. The Art of Software Testing. Wiley Series in Business Data Processing, ed: R. Canning, J.D. Couger, 1979.

[SKPO88] M. Stonebraker, R. Katz, D. Patterson, J. Ousterhout. The Design of XPRS. *Proc. 14th International Conference on Very Large Data Bases*, August, 1988.

[Sto91] M. Stonebraker. The Case for Shared Nothing. *IEEE Database Engineering*, March, 1991.

[TDG87] Tandem Database Group. NonStop SQL: A Distributed High Performance, High Availability Implementation of SQL. *High*

Performance Transaction Systems, Berlin: Springer-Verlag, 1987.

[TPG88] Tandem Performance Group. A Benchmark of NonStop SQL on the Debit Credit Transactions. *Proc. ACM SIGMOD conference*, June, 1988.

[Wil92] D. Wildfogel. Dynamic and Exploratory Testing. *Proc. 5th International Software Quality Week*, May, 1992.

[Wil91] D. Wildfogel. NAPR: The Testing Environment for Low Level Database QA. *Proc. TSG Productivity and Quality Conference*, October, 1991.