Dependability Benchmarking of Linux based Systems

Oliver Tschäche

Institut für Informatik 3
Friedrich Alexander Universität Erlangen-Nürnberg
Martensstr. 3
D-91058 Erlangen
ot@cs.fau.de

Abstract:

Known benchmarks, like SPECweb, SPECint or TPC, evaluate system performance. In general, they evaluate characteristics like speed, response time, usage of resources etc. They usually ignore issues of reliability and dependability.

This paper shows a way to extend these performance benchmarks by dependability aspects. It addresses the issues: How does system performance change facing hardware faults? How can we evaluate measures for these performance changes?

An example of a dependability benchmark for database systems exercises the methodology of extending a well known benchmark, the TPC Benchmark™C, to a dependability benchmark for online transaction processing.

At last the implementation of that dependability benchmark shows the operational accuracy of the approach. FAUmachine, a fast PC-hardware emulator with the ability of fault injection, supported by a faultload generator is used to stress an Oracle database server running on that virtual Linux system. The results are performance measures showing the loss of performance in presence of hardware faults.

1 Introduction

Benchmarks are used to compare systems, often as marketing instruments. They tell customers which system offers the best ratio of performance and price. Performance is defined as speed, delay, usage of resources etc. The left side of figure 1 (dark grey shaded) shows the classical setup of a performance benchmark:

- SUT: The System Under Test is the benchmarking target. A detailed description of the system (number of CPUs with their speed, memory size, network connections and so on) has to be included to the benchmarking results in form of a full disclosure report.
- WL: The WorkLoad describes what has to be done by the SUT. The WL is completely defined in the benchmark. The question of how the SUT handles the WL is implementation dependent. Again, all implementation details have to be layed open in a full disclosure report of the results of the benchmark.

- PM: The Performance Measures are the result of running the WL on the SUT. They include informations about e.g. the speed, delays and/or resource usage etc. These measures are the numbers used as marketing instruments.

Performance benchmarks miss dependability aspects. To compare reliability and availability of different systems, a performance benchmark needs to integrate these aspects into a benchmarking process, too. This paper shows an approach to extend performance benchmarks to dependability benchmarks, which are presented in section 2.

For the proof of concept the approach is exercised on the well known TPC Benchmark ™C (TPC-C). TPC-C is a popular yardstick for comparing online transaction processing performance on various hardware and software configurations. TPC-C simulates a complete computing environment where a population of users executes transactions against a database. A short summary of the extensions applied to TPC-C creating TPC-C DBench is shown in section 3.

An implementation of TPC-C DBench is shown in section 4. Benchmarking target is an Oracle database server running on the Linux operating system. An example of measures for that approach is shown in section 5.

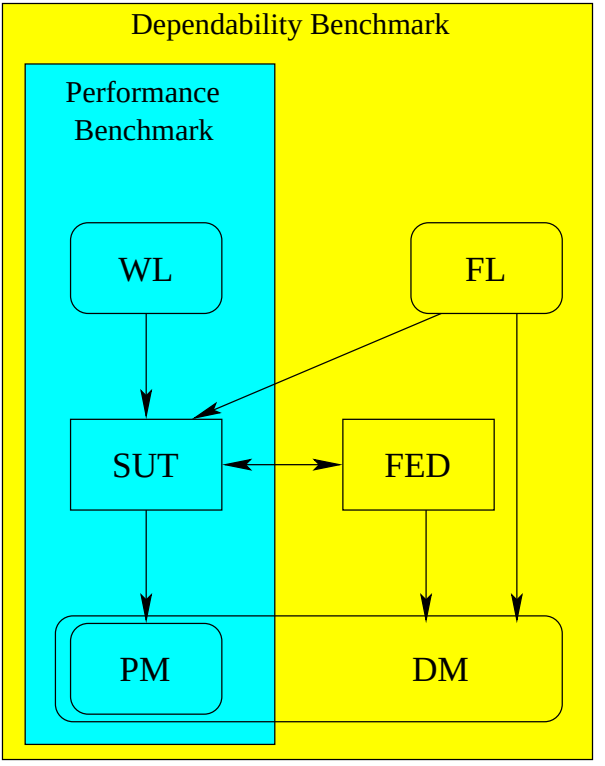

Figure 1: Dependability Benchmark Setup

2 Dependability Benchmarking

This section defines requirements for a dependability benchmark. The goal of dependability benchmarking is to evaluate availability and reliability aspects of a system. To evaluate these aspects the system is stressed with faults in addition to the workload. The questions are:

- Which effect does a fault have in the system?
- Which dependancies between fault effects and faults, workload, time of fault activation etc. can be observed?
- How does the system perform in the presence of faults?

An effective dependability benchmark should at least answer these questions.

The last question ”How does the system perform in the presence of faults?” targets evaluation of parameters which are already covered by performance benchmarks for fault free systems. Therefore, dependability benchmarking is seen as an extension to performance benchmarking by this approach. The right side of figure 1 shows the dependability extensions to the performance benchmark:

- FL: The FaultLoad stresses the SUT with faults in addition to the WL. A set of addressable hardware faults of our implementation is shown in section 2.2. For a methodology of how to apply faults, see section 2.1.
- FED: The Fault Effect Detector analyses the behaviour of the SUT after a fault was activated and classifies the behaviour into fault effects, see section 2.3 for a list of these effects.
- DM: The dependability measures are a combination between PM and the detected fault effect. DMs are statistics including the distribution of fault effects to each fault, see section 2.4.

According to the overview in figure 1 the following sections provide an insight to the dependability benchmark elements.

2.1 Running a Dependability Benchmark

In case of performance benchmarks the WL is applied to the SUT once in a single run. In that run all performance informations are evaluated. Dependability benchmarks, too, will include a fault free run as a reference. In this paper this run will be referred to as ”Golden Run”, see [STB97].

The following items specify requirements which are met by our approach. They explain how to run the WL and FL doing dependability benchmarking instead of performance benchmarking:

- In addition to the Golden Run other runs must evaluate the system performance in presence of a single or more faults. This run is called an experiment. For a set of addressable faults see section 2.2. An experiment runs the same duration as the Golden Run. This duration has to be documented in a full disclosure report.
- A fault must be selected randomly out of the set of faults, see section 2.2. The distribution of the different kinds of faults has to be layed open in a full disclosure report. A general guide to choose faults for the experiments is to look at their rates of appearance. E.g. if the frequency of disk faults is twice the frequency of power fail faults then the number of experiments with activated disk faults should be twice the number of experiments with activated power fail faults, see [Si98].
- For an experiment a fault must not be activated before the SUT is ready to perform the WL. This is to fault protect the time a system needs to reach the state being able to handle the WL. This is fair in respect to the Golden Run, for which the performance measures do not start before that point of time. That point of time must be defined and included in a full disclosure report.
- For each experiment a fault must be deactivated at least a minimum time before the WL finishes. That interval of time is called End Of Fault Activation (EOFA) and has to be reported in the full disclosure report. SUTs with fault detection and recovery mechanisms may use this time to recover from an erronous system state. The FED element, see section 2.3, has to detect whether the SUT was successful in recovering from that state.
- The time at which a fault is activated must be uniformly distributed between the start of applying the WL and EOAF before the end of the experiment.
- The duration a fault is activated must be exponential distributed around an expected value, described in the full disclosure report.
- The benchmark setup must be unambigous specified, so that anybody may reproduce the results, see [BCH+03]. Therefore, the setup must be specified in a machine readable format.
- To reduce the time needed to compute DMs, it is allowed to run experiments in parallel distributed on several systems. But it must be provided that the hardware and software configuration of the benchmarking environment is identical.

2.2 Faultload (FL)

Dependability benchmarking extends performance benchmarking to aspects of availability and reliability. Or - to put it in another way - dependability benchmarking evaluates system performance in the presence of faults.

The remaining question is, which faults should be addressed? There are a wide variety of faults: Operating system faults, operator faults, software faults, hardware etc. Section 2.1 is independent of the fault type and addresses all faults.

This approach addresses the following high level hardware faults, which are used to transform a performance benchmark into a dependability benchmark:

- Power fail fault: Drop out of the powersupply.
- Harddisk faults: According to the harddisk geometry it is differentiated between following subtypes:
  - Defect disk: Reading or writing to the affected disk must result in a read/write error.
  - Defect head: Reading or writing to the set of blocks accessed by this head must result in a read/write error. Access of other blocks works properly.
  - Defect track: Reading or writing to the set of blocks belonging to a single track must result in a read/write error. Access of other blocks works properly.
  - Defect block: Reading or writing to that single block must result in a read/write error. Access of other blocks works properly.
- Network faults: There are two types of network faults: Receive and send faults. This fault makes an incoming or outgoing network package got lost. The fault is activated by setting the probability for losing a package. This probability is called loss rate.
- Memory faults: A single bit of the memory flips to the other state.
- CPU fault: A single bit of a register of the CPU flips to the other state. This fault is dependent on the hardware architecture.

2.3 Fault Effect Detector (FED)

A dependability benchmark has to analyze the system state after each experiment to find out which effect a fault had. In contrast to the Golden Run an experiment may contain erronous calculated data. Imagine a benchmark working on the forecast of weather with a fault in the arithmetic unit of some processors. Where the Golden Run forecasts sunshine an experiment may forecast rain.

Therefore, the results of the experiments have to be classified into fault effects. Facing our approach of this dependability benchmark the results must be distinguished between the following groups:

- PR, Perfect Run: The experiment acted like the Golden Run. Full performance without any errors.
- GD, Graceful Degradation: The data is correct, but the system performance reduces.
- SDW, SUT working: The SUT detected an inconsistency itself, reported an error and keeps working.

- SDS, SUT shutdown: The SUT detected an inconsistency itself, reported an error and shuts down.
- SCR, SUT crashes restartable: The SUT crashes but is available after restart.
- SCM, SUT crashes and needs maintanance: The SUT crashes and does not restart after reset but needs maintanance.
- SW, SUT working: The SUT keeps working with undetected data inconsistency.
- ND, not determined: The SUT's state can't be classified into one of the above states.

In addition to the WL and FL a Fault Effect Detector (FED) must be implemented. The duty of the FED element is to classify the experiments into the groups of fault effects listed above.

The FED must not interfere with the SUT during an experiment. Therefore, one way to implement the FED is to analyze the SUT's state after an experiment has finished, which means after the PMs are taken.

2.4 Dependability Measures (DM)

The dependability measures are two statistics over all experiments related to the fault effects detected by the FED. The statistics have to be in the following decoupled formats:

- The effect statistic includes a list of all detected fault effects coming along with the percentage of faults creating that effect. This fraction is the experimental determined probability, that any fault triggers that effect.
- The fault statistic includes a list of all injected fault types, see section 2.2, coming along with the distribution of fault effects, each one produces. These fractions are the experimental determined probabilities of a fault leading to a fault effect.

For performance benchmarking the measures tell by hard numbers which system is the better one (according to that benchmark). With the proposed methodology statistics are generated. These are based on experiments with randomly injected faults. Therefore, each experimental determined probability must come along with a confidence interval explaining the degree of reliance for it. The more tighten this interval is the more confident and the more sharp is the measure. The size of the confidence interval must be calculated by equation 1, which is derived from Tschebyscheff's inequality which assumes the worst distribution. According to equation 1 the size depends on the number of experiments n and α, containing the probability that the measured probability lies outside that confidence interval. $1 - \alpha$ is the probability for being right with the measured value. $1 - \alpha$ is the degree of reliance. The term $t_{\alpha,n-1}$ is the term for the Student t-distribution. Some values for $t_{\alpha,n-1}$ with $n \to \infty$ are shown in the following table. These values may be used to approximate the size of the confidence interval if $n \gg 100$ experiments:

α	0.100	0.010	0.001
$t_{\alpha,n\to\infty}$	1.645	2.576	3.291

The selected degree of reliance has to be included in the full disclosure report. The interval can be sharpened by doing more experiments. If two systems are compared, a statement of quality dependent on the measures is only valid, if the intervals of these measures do not overlap. If the intervals overlap the statement of quality hides in statistical noise and is defined invalid by this approach.

$$size(\alpha, n) = \frac{t_{\alpha,n-1}}{\sqrt{n-1}} \tag{1}$$

Up to here no pricing is included in these values. Applying the approach of this dependability benchmark includes, that the pricing must be layed open in the full disclosure report. For each fault effect a worst case price must be estimated. E.g. a bank has huge interest to recognize a bad transaction on any account. Comparing systems by pricing is only valid if calculated by applying the confidence intervals as stated before. The comparison is only valid if pricing intervals do not overlap.

3 TPC-C DBench

This section shortly summarizes how the methodology of section 2 is applied to the well known TPC Benchmark™C (TPC-C), [TPPC01]. This extends TPC-C from a performance benchmark to a dependability benchmark.

TPC-C is a benchmark defined by the Transaction Processing Performance Council (TPC). TPC "defines transaction processing and database benchmarks, which deliver trusted results to the industry" [TPPC01]. For the main subject, TPC-C version 5.0 defines on 128 pages a workload (WL), a set of performance measures (PM) and the properties of a database system (SUT) processing the WL. TPC-C is a benchmark as shown on the left, dark grey shaded side of figure 1.

Section 2 addresses the dependability extensions, as shown on the right side of figure 1. Because the FL is only hardware dependent and not software dependent and because TPC-C does not have any hardware requirements, TPC-C can be easily extended with it by adding a new clause including the targets of section 2.2. Additional requirements might look like a restriction of TPC-C. But because the FL is described at an abstract level and therefore can be applied to any computer system, adding a new clause is an extension and not a restriction of TPC-C.

The FED is more complex: TPC-C's workload is modelled with a lot of remote terminals connected to the database system. At each terminal a remote terminal emulator creates a randomly produced user interaction. During an experiment slight delays produced by interrupts, or different access times of the harddisk etc. lead to different sequences of database transfers because the transactions at the terminals are encountered at different times. This complicates automatic detection of differences in the state of the database

of the Golden Run and the experiment. Thus, a simple comparison of the database state after an experiment will end in pretendedly incorrect data, although the data is correct. Therefore, the FED needs to use more sophisticated comparison methods, see section 4.

The DMs of section 2.4 only depend on the FED and include the PMs for each experiment. Thus, these rules can be applied to TPC-C in the same way as the FL by adding an additional clause.

4 Implementation of TPC-C DBench

This section introduces an approach to implement the proposed TPC-C DBench of section 3. This implementation is based on three major blocks, see section 4.1 ff. These blocks cover the whole scenario introduced by figure 1.

Running the experiments is done fully automatically. This is extremely useful because of the large number of experiments (more than 1000) needed to get results of high confidence. Running an experiment and evaluating the results are two tasks in our approach. Setting up the SUT and applying the WL and FL during an experiment is done by the load generator. The load generator is connected to the SUT. The SUT is a Hardware Simulator which is startet from the Load Generator as soon as it has setup the SUT. When the Load Generator detects the end of the WL, it terminates the SUT. Then the FED analysis starts.

4.1 Hardware Simulator

There are several methods to inject faults. This implementation chooses the way of simulating hardware. This means, the SUT, consisting mainly of the database and its operating system and the hardware, is not running on real hardware but is running on a simulated, virtual hardware.

FAUmachine, earlier called UMLinux, [HBS02], [SB02], [FA03] is an approach implementing a high performance hardware simulation of Intel 80x86 based systems. FAUmachine requires that the operating system running on the virtual hardware is Linux. In the virtual Linux-machine it is nearly impossible for an application to recognize, that it is running in a virtual machine. Because Oracle database server is certified for SuSE-Linux, SuSE-Linux was choosen to be the operating system of the virtual machine.

Because FAUmachine simulates hardware, hardware faults arrive first at the Linux driver of the operating system and are passed to the application as errors, erronous data, pretended correct data or as correct data (if the hardware or operating system is capable of fault tolerance for this fault, e.g. harddisk faults in case of RAID). FAUmachine supports the ability to inject all hardware faults listed in section 2.2.

TPC-C defines an application server translating between user, who fills in forms, and the Oracle database server, which handles SQL-statements. Each form a user submits to the application server is passed as SQL-statements to the Oracle database server. The ap-

plication server has been implemented by [Ka03]. The application server uses the serial interface of a virtual machine to communicate with the user connected by a serial terminal. The load generator is connected to the serial interface. The application server is running like the Oracle database on a virtual Linux system.

For this approach, the application server applies the transaction on two databases: The first is part of the SUT and the second serves as reference database for the FED.

4.2 Load Generator

The Load Generator has two tasks: First set up the SUT and, then, start the SUT and run the WL and FL. Because the SUT consists of virtual hardware, the setup of the initial state of a virtual machine is just to create the virtual harddisk image. This is done by copying a preinstalled disk image which contains the initial state.

Starting the SUT means to start the Linux Simulator and connecting the Load Generator to the virtual serial interfaces of the SUT.

TPC-C exactly defines a WL modelling user interaction at each terminal. The WL is implemented in VHDL [Ka03]. VHDL is a programming language to describe parallel processes. We use it to model user input at the serial terminals. A VHDL-interpreter is connected to FAUmachine and creates the load at the serial terminals of the virtual system.

4.3 FED Analysis

FED analysis examines the effect of a fault to the SUT and protocols it to the DMs. For this approach the FED Analysis compares the SUT's database state against the reference database during experiment runs after each submitted transaction. In addition system log files are searched for errors which were generated by the operating system or the database server.

The implementation of a reference database was chosen to get around the problem described in section 3. This breaks with the dependability benchmark requirement that the FED must not interfere with the SUT. For this approach it is defined that only the Oracle database server is part of the SUT and not the application server, which does additional logging for FED purposes. Because this implementation variant is a restriction for TPC-C future implementations will be implemented more sophisticated without breaking with the requirement of interference.

Classifying fault effects is done by the following investigations:

- PR: No error encountered by comparison of transfers to the refence database nor any log message from the operating system and database server.
- GD: Like PR but less throughput of transactions. E.g. in case of network faults, the TCP/IP stack of the operating system retransmits lost packages.

- SDW: Operating system or database server complains about errors, but keeps working. E.g. the operating system logs read error of a harddisk partition, which is a member of a softraid partition. Or the database server logs, that it has rejected a transmission after it tried to apply it to the database.
- SDS: Encountered if the database server logs disk access faults and shuts down.
- SCR: After power fail fault the system restarts with successful filesystem check.
- SCM: After power fail fault the system restarts but needs maintanance of a system administrator to do the filesystem check.
- SW: No log messages registered but comparison of databases differs.
- ND: The SUT's operating system or the database server issues an error message, which does not match any FED's pattern. This effect needs manual investigation of the experiment's data. It can be used to extend patterns of the above mentioned classes at the development phase of the FED.

Because the goal of dependability benchmarking is comparing systems, the FED of the Oracle based system should be the same as the FED of an e.g. Informix based system. This can't be managed because each database server system writes different error messages. Thus, the FED must be implemented very accurately and, finally, the FED implementations should be verified against each other, before they are used to evaluate system performance.

5 Results

More detailed demonstration of the DMs evaluated with this approach can be found in [BCH+03]. Detailed results are demonstrated in [Ka03]. An example of network faults is demonstrated in figure 2. The percentage of network loss is assigned to the horizontal axis. TPC-C suggests the unit tpmC meaning transactions per minute of TPC-C. This value is assigned to the vertical axis. In case of network faults the fault effect was GD independent of the loss rate.

Figure 2 visualizes the number of transactions finished per minute. There are two curves derived from two faults: send and receive loss of network packages. The fault effect of both faults was always graceful degradation. The curves show that the number of transactions decreases the higher the loss rate is. The vertical intervals around the curves describe the confidence interval of the measures.

6 Conclusion

This paper presents a methodolgy to extend performance benchmarks to dependability benchmarks. It exercises this approach to the well known TPC Benchmark ™C creating a

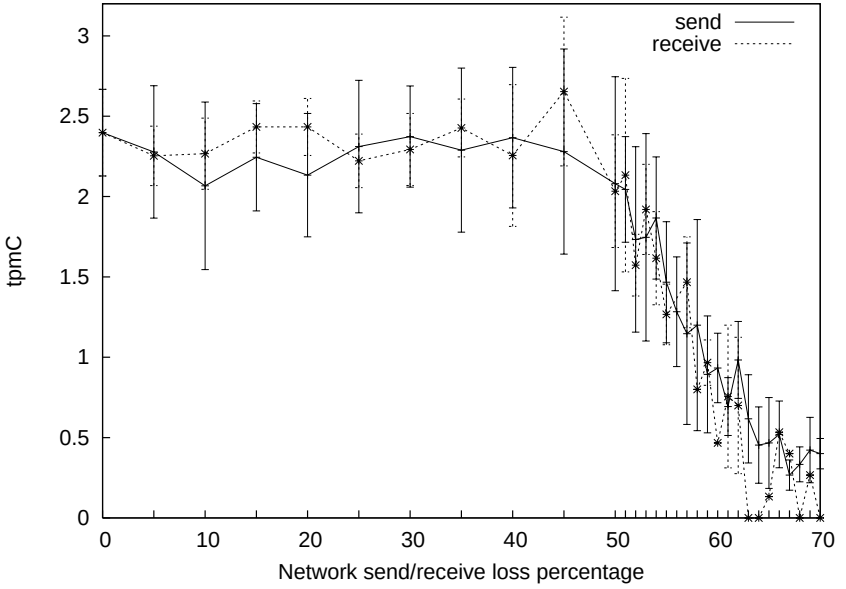

Figure 2: Performance of TPC-C dependent on probability for Network Packet Losses

dependability benchmark for database systems.

This paper introduces an approach to implement dependability benchmarks for Linux based Intel systems with FAUmachine supported by a Load Generator being able to interpret VHDL code. Thus, the WL and FL has to be modelled in VHDL and the SUT must be running Linux operating system.

The hardest part of implementing this approach is the FED analysis. The more complex the SUT is the more difficult it will be to find the patterns which are used to classify system behaviour into fault effects for each fault. In addition, these patterns are highly dependent on the SUT's application and, therefore, conflict with the suggested rule of reproducability.

Transfering the DMs evaluated with this approach to real systems is questionable. The effects of variations of timing between the real system and our simulation based approach can't be estimated yet. Thus, the DMs are only valid for running the SUT in that virtual environment.

7 Outlook

Future implementation of TPC-C DBench will resolve the conflict with the requirement of interference. Therefore, more sophisticated FED methods must be found and implemented.

More efforts will be spent to find methods implementing FEDs more accurately facing

different applications. A modular implemented FED extracting common parts of different applications. E.g. comparing Linux based Oracle and Linux based Informix can use same modules for operating system dependent investigation. Because there will always be partly different FED moduls for different systems, methods must be found to verify that these moduls do the same classification of fault effects.

Implementing the fault injector in hardware would lead to measures being more comparable to real hardware. But similar to Heissenberg postulation: The more accurate you want to measure the less noticable (for the system) must be the intrusion of your observer.

Acknowledgement

The research presented in this paper is supported by the European Community (DBench project, IST-2000-25425). We want to thank all the other people who contributed to our benchmarking environment FAUmachine.

References

[BCH+03] Buchacker, K., Cin, M. D., Höxer, H.-J., Karch, R., Sieh, V., und Tschäche, O.: Reproducible dependability benchmarking experiments based on unambiguous benchmark setup description. In: *Dependable Systems and Networks*. S. 469–478. 2003.

[FA03] FAUmachine Team. FAUmachine. URL: http://www.FAUmachine.org/. 2003.

[HBS02] Höxer, H., Buchacker, K., und Sieh, V.: UMLinux - a tool for testing a linux system's fault tolerance. In: *LinuxTag 2002, Karlsruhe, Germany, June 6-9*. 2002.

[Ka03] Karch, R.: Implementation and analysis of a dependability benchmark for the oracle-, informix- and postgresql database servers. Master's thesis. University of Erlangen, Computer Architecture. 2003.

[SB02] Sieh, V. und Buchacker, K.: UMLinux — a versatile SWIFI tool. In: Bondavalli, A. und Thevenod-Fosse, P. (Hrsg.), *Fourth European Dependable Computing Conference, Toulouse, France, October 23-25, 2002*. S. 159–171. Springer Verlag, Berlin. 2002.

[Si98] Sieh, V.: *Effiziente Erstellung und Auswertung von Rechnermodellen zur detaillierten Zuverlssigkeitsanalyse*. PhD thesis. University of Erlangen, Computer Architecture. 1998.

[STB97] Sieh, V., Tschäche, O., und Balbach, F.: VERIFY: Evaluation of reliability using VHDL-models with integrated fault descriptions. In: *Proceedings of the 27th IEEE International Symposium on Fault Tolerant Computing*. S. 32–36. 1997.

[TPPC01] Transaction Processing Performance Council. TPC Benchmark [tm] C, Standard Specification, Revision 5.0, February 26, 2001. www.tpc.org. 2001.