

TPC Benchmark™C Version 5.2 Dependability  
Benchmark Extensions  
(*TPC-C Depend Extensions* )  
Draft Version 1.0

Institut für Informatik 3  
Friedrich Alexander Universität Erlangen-Nürnberg  
Germany

February 25, 2004

## **Acknowledgements**

The European Union provided funding for this work through the DBench Project (IST-2000-25425) [1].

## **Purpose**

This document is based on the well known TPC Benchmark™C Version 5.2 (TPC-C) [2] for Online Transaction Processing (OLTP) Systems. The TPC-C is published by the Transaction Processing Performance Council (TPC, *www.tpc.org*).

This document extends the TPC-C into the dependability domain. That is, it extends or modifies clauses or adds new clauses to the TPC-C, in such a way, that the resulting document not only covers benchmarks performance aspects of a system but also dependability aspects.

Any clause not listed explicitly in this document remains unmodified and is taken from TPC-C [2].

# Contents

|  |           |
|--|-----------|
| <b>0 Preamble</b>                                  | <b>5</b>  |
| 0.1 Introduction . . . . .                         | 5         |
| 0.2 General Implementation Guidelines . . . . .    | 6         |
| <b>1 Logical Database Design</b>                   | <b>7</b>  |
| <b>2 Transaction and Terminal Profiles</b>         | <b>9</b>  |
| <b>3 Transaction and System Properties</b>         | <b>11</b> |
| 3.2 Atomicity Requirements . . . . .               | 11        |
| 3.3 Consistency Requirements . . . . .             | 11        |
| 3.4 Isolation Requirements . . . . .               | 11        |
| 3.5 Durability Requirements . . . . .              | 11        |
| <b>5 Metrics</b>                                   | <b>13</b> |
| 5.1 Definition of Terms . . . . .                  | 13        |
| 5.5 Measurement Interval Requirements . . . . .    | 15        |
| 5.6 Required Reporting . . . . .                   | 15        |
| 5.7 Primary Metrics . . . . .                      | 17        |
| 5.8 Failure Modes . . . . .                        | 17        |
| <b>6 SUT, Driver and Communications Definition</b> | <b>21</b> |
| 6.3 System Under Test (SUT) Definition . . . . .   | 21        |
| <b>7 Pricing</b>                                   | <b>23</b> |
| 7.3 Maintenance . . . . .                          | 23        |
| 7.4 Required Reporting . . . . .                   | 23        |
| <b>8 Full Disclosure</b>                           | <b>25</b> |
| 8.1 Full Disclosure Report Requirements . . . . .  | 25        |

|                                       |           |
|---------------------------------------|-----------|
| <b>9 Audit</b>                        | <b>27</b> |
| <b>10 Faultload (FL)</b>              | <b>29</b> |
| 10.1 Fault Mix . . . . .              | 29        |
| 10.2 Required Reporting . . . . .     | 29        |
| 10.3 Fault Recovery Pricing . . . . . | 30        |
| <b>D Sample Faults</b>                | <b>31</b> |
| D.1 Sample Disk Faults . . . . .      | 31        |
| D.2 Sample Network Faults . . . . .   | 31        |
| D.3 Sample Processor Faults . . . . . | 32        |
| D.4 Sample Memory Faults . . . . .    | 32        |

# Clause 0

## Preamble

Clause 0.1 of TPC-C [2] is modified.

### 0.1 Introduction

The following statements are added to clause 0.1 of TPC-C [2].

Large transactional systems are usually at the very center the IT infrastructure of companies, banks and other institutions, handling huge amounts of data every day. Even short downtimes of such a system are very expensive. Unexpected losses of important data may even put a company out of business. Clearly, both performance and dependability benchmarks are very important to select the right scale and type of transactional system for the job.

Performance benchmarks for transactional systems have been around for a long time. The de facto standards, which are accepted and supported by database, operating system and hardware vendors alike are the benchmarks from the Transaction Processing Performance Council (TPC, [www.tpc.org](http://www.tpc.org)). There are a number of different TPC benchmarks for different types of database systems. The current performance benchmark for large databases in a client-server or three-tier environment is the TPC Benchmark™C Version 5.2 (TPC-C) [2]. We have therefore based our research on the TPC-C. The TPC-C is a document, specifying how the benchmark must be implemented. The TPC ensures conformance with the specification before the benchmarks results may be published. Implementation details must be laid open in a full disclosure report, so that independent parties can reproduce the benchmark.

The TPC-C performance benchmark covers three main aspects. It defines a workload and a set of performance measures for database systems, as well as rules concerning full disclosure of the setup for certain benchmark results. The aim of the TPC-C is to make it possible to compare database systems fairly. The rules for full disclosure are important to determine, whether the performance delivered by two different systems having been benchmarked can indeed be compared directly, and to make benchmark results reproducible.

*TPC-C Depend Extensions* extends TPC-C to a benchmark including dependability aspects. *TPC-C Depend Extensions* defines a faultload in addition to the workload already defined by TPC-C. The performance measures defined by TPC-C must be accompanied by dependability measures in *TPC-C Depend Extensions* and the full disclosure

rules must be updated accordingly. Only the shaded part in figure 1 is defined in this document. This document has the same structure as the TPC-C [2] and will often refer to clauses within the TPC-C. To obtain a dependability benchmark for OLTP systems based on TPC-C, this document must be combined with TPC-C as is shown in figure 1. This document by itself does not specify a complete dependability benchmark!

## 0.2 General Implementation Guidelines

The following statements are added to clause 0.2 of TPC-C [2]:

To achieve reproducibility of the measured results, the complete benchmarking setup including hardware, software, workload, faultload, etc. must be defined unambiguously. This is usually done in the full disclosure report.

True unambiguity, however, can only be achieved if the complete benchmarking setup is described in a machine readable language with well defined syntax and *semantics*. In this document, we therefore recommend publishing this machine-readable benchmarking setup as part of the full disclosure report.

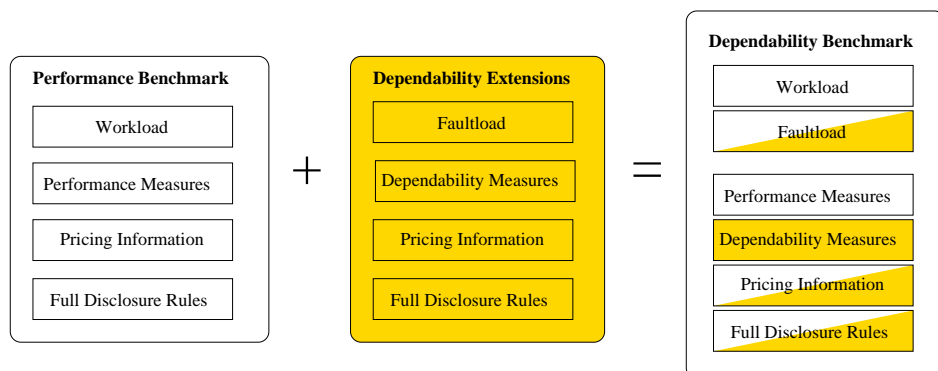


Figure 1: Equation of Dependability Benchmark

## **Clause 1**

# **Logical Database Design**

This clause remains identical to clause 1 of TPC-C [2].



## **Clause 2**

# **Transaction and Terminal Profiles**

This clause remains identical to clause 2 of TPC-C [2].



## **Clause 3**

# **Transaction and System Properties**

Clause 3 of TPC-C [2] is modified.

### **3.2 Atomicity Requirements**

The following statements are added to clause 3.2 of TPC-C [2].

Atomicity requirements need only be fulfilled during SUT behaviour classified as "fully functional".

### **3.3 Consistency Requirements**

The following statements are added to clause 3.3 of TPC-C [2].

Consistency requirements need only be fulfilled during SUT behaviour classified as "fully functional".

### **3.4 Isolation Requirements**

The following statements are added to clause 3.4 of TPC-C [2].

Isolation requirements need only be fulfilled during SUT behaviour classified as "fully functional".

### **3.5 Durability Requirements**

The following statements are added to clause 3.4 of TPC-C [2].

Durability requirements need only be fulfilled during SUT behaviour classified as "fully functional".



## Clause 5

# Performance and Dependability Metrics and Response Time

Clause 5 of TPC-C [2] is modified.

### 5.1 Definition of Terms

The following statements are added to clause 5.1 of TPC-C [2]:

#### 5.1.3 Faultload

This clause is new.

The **faultload** consists of  $N$  single faults  $f_i$ .

#### 5.1.4 Failure Modes

This clause is new.

The term **failure mode** refers to a subset of possible observable behaviors of the SUT. Let the full set of possible behaviors of the SUT be called  $\Omega$ . Failure modes must partition  $\Omega$ . I.e. no two failure modes may overlap and the union of all failure modes must be  $\Omega$ . Failure modes classes are described using conditional expressions.

Refer to clause 5.8 for a list of failure modes and their conditional expressions.

#### 5.1.5 Available and Unavailable Set

The *available* set  $S_A$  contains all those failure modes corresponding to availability of the system. It must contain the *fully functional* failure mode.

The *unavailable* set  $S_U$  contains all those failure modes corresponding to unavailability of the system. It must contain the *unknown* failure mode.

$$FM_0 \in S_A \quad (5.1)$$

$$FM_M \in S_U \quad (5.2)$$

### 5.1.6 Stationary Availability

Availability is a measure for the ability of a system to be functional at a certain point in time. It is a measure used for repairable systems. The current availability  $A(t)$  depends on variables such as the knowledge of the repair technicians, logistic support, reparability of the system. These variables are usually hard to quantify, which makes  $A(t)$  very hard to calculate. The stationary availability  $A$  is the limit of  $A(t)$  for  $t \rightarrow \infty$ . Assuming that a system provides continuous service and is repaired when a failure is detected, we can calculate  $A$  as

$$A = \frac{T_{\text{up}}}{T_{\text{total}}} \quad (5.3)$$

$T_{\text{up}}$  is the uptime of the system, during which it is available.  $T_{\text{total}}$  is the total time the system is observed.

### 5.1.7 Stationary Unavailability

The stationary unavailability can be calculated as  $U = 1 - A$ .

### 5.1.8 Experiment

This clause is new.

The term **experiment** refers to a single benchmark execution, with or without faultload (see also clause 10 in this document).

An experiment with faultload determines the effects of a single fault.

### 5.1.9 Golden Run

A *Golden Run* is an experiment without faultload. **Comment:** The Golden Run is equivalent to execution of the performance benchmark without the dependability extensions. The Golden Run serves as reference for the other runs with activated faultload.

### 5.1.10 Dependability Benchmark

This clause is new.

For this benchmark to qualify as a **dependability benchmark**, a number of experiments (as defined in clause 5.1.8) with the following properties must be conducted:

1. There must be at least one experiment without faultload (Golden Run).
2. An experiment with faultload gathers information used to calculate the relative frequency of a certain failure mode with a certain confidence interval. The behaviour of the system under benchmarking observed during the experiment must be classified into a failure mode.
3. The number of experiments must be high enough to derive confidence intervals of 90% or more for the occurrence of the failure mode classes.

4. At the start of each experiment, the system under benchmarking must be in the same state.

**Comment:** TPC-C runs the workload one time. All measures of TPC-C are derived from that single run. *TPC-C Depend Extensions* needs more than a single run to derive its dependability measures. At first a *Golden Run* derives measures without any faultload. The Golden Run is equivalent to execution of the TPC-C without the dependability extensions. The Golden Run serves as reference for the other runs with activated faultload, called experiment. An experiment determines the effect of one single fault, see chapter 5.1.

## 5.5 Measurement Interval Requirements

The following statements are added to clause 5.5 of TPC-C [2]:

### 5.5.3 Timing of Faults

This clause is new.

The earliest time to inject a fault is then, when the SUT is ready to handle the workload and the workload starts. The fault must be activated randomly but uniformly distributed after that time and 5 minutes before the experiment finishes. **Comment:** Thus, the SUT has 5 minutes to recover from an encountered error.

## 5.6 Required Reporting

The following statements are added to clause 5.6 of TPC-C [2]:

### 5.6.5

This clause is new.

The total number  $n$  of experiments must be reported.

### 5.6.6

This clause is new.

The relative frequencies and confidence intervals for the occurrence of the failure modes must be reported in in the form of a table (see table 5.1 for an example). The column-headings are the failure modes  $FM_j$  (refer to clause 5.1.4 for a definition and clause 5.8.1 for a list). The row-headings are the single faults  $f_i$  which constitute the faultload. There are  $M$  different failure modes and  $N$  different faults in the faultload.  $FM_0$  is the *fully functional* failure mode.  $FM_M$  is the *unknown* failure mode.

| Fault | Failure Mode |          |     |          |     |          |
|-------|--------------|----------|-----|----------|-----|----------|
|       | $FM_0$       | $FM_1$   | ... | $FM_j$   | ... | $FM_M$   |
| $f_0$ | $h_{00}$     | $h_{01}$ | ... | $h_{0j}$ | ... | $h_{0M}$ |
| $f_1$ | $h_{10}$     | $h_{11}$ | ... | $h_{1j}$ | ... | $h_{1M}$ |
| ...   | ...          | ...      | ... | ...      | ... | ...      |
| $f_i$ | $h_{i0}$     | $h_{i1}$ | ... | $h_{ij}$ | ... | $h_{iM}$ |
| ...   | ...          | ...      | ... | ...      | ... | ...      |
| $f_N$ | $h_{N0}$     | $h_{N1}$ | ... | $h_{Nj}$ | ... | $h_{NM}$ |

Table 5.1: Failure Mode Table

### 5.6.7

This clause is new.

The cost  $C_j$  associated with a failure mode  $FM_j$  must be reported. **Comment:** There is also a cost  $C_j$  associated with each failure mode  $FM_j$ . If the failure mode entails loss of data,  $C_j$  could include the cost to restore and older version of the lost data from a previous backup as well as the loss of customers due to the loss of data. If a failure mode is a reduced performance mode, the cost would be due to the loss of online business, because of the sluggish response of the server to online requests.

### 5.6.8

This clause is new.

The occurrence rates  $R_j$  of all failure modes  $FM_j$  used (or Mean Time Between Occurrence ( $MTBO_j = 1/R_j$ ) instead).

$R_j$  can be calculated from the information in table 5.1 if the fault rates  $r_i$  are known for all faults part of the faultload:

$$R_j = \sum_{i=0}^N (r_i \cdot h_{ij}) \quad (5.4)$$

**Comment:** Because we should treat the relative frequencies  $h_{ij}$  as random variables, we should use the Chebyshev equation to calculate a confidence interval for  $R_j$ .

### 5.6.9

This clause is new.

The repair rates  $Q_j$  of all failure modes  $FM_j$  used (or Mean Time To Repair ( $MTTR_j = 1/Q_j$ ) instead).

$Q_j$  can be calculated from the information in table 5.1 if the repair rates  $q_i$  are known for all faults part of the faultload. The repair rate can be calculated from the mean time

to repair (MTTR) as  $q_i = 1/MTTR$ .

$$Q_j = \sum_{i=0}^N (q_i \cdot h_{ij}) \quad (5.5)$$

**Comment:** Because we should treat the relative frequencies  $h_{ij}$  as random variables, we should use the Chebyshev equation to calculate a confidence interval for  $Q_j$ .

## 5.7 Primary Metrics

Clause 5.7 of TPC-C [2] is modified.

### 5.7.1

The following statements are added to clause 5.7.1 of TPC-C [2]:

- The occurrence rates  $R_j$  of all failure modes used (or MTBO instead).
- The repair rates  $Q_j$  of all failure modes used (or MTTR instead).
- $S_A$ , the available set, and  $S_U$ , the unavailable set.
- The stationary availability:

$$A = \frac{\sum_{FM_j \in S_A} MTBO_j}{\sum_{FM_j \in S_A \cup S_U} (MTBO_j + MTTR_j)} \quad (5.6)$$

## 5.8 Failure Modes

This clause is new.

The term **failure mode** refers to a subset of possible observable behaviors of the SUT. Let the full set of possible behaviors of the SUT be called  $\Omega$ . Failure modes must partition  $\Omega$ . I.e. no two failure modes may overlap and the union of all failure modes must be  $\Omega$ . Failure modes are described using conditional expressions.

**Comment:** This document covers hardware failures only and the failure modes are chosen accordingly.

### 5.8.1 Definition of Fault Effect Classes

The following list describes the conditions which must hold for a certain SUT behavior to be assigned into a certain fault effect class.

**Fully functional (FF):** The following conditions must hold:

The injected fault had no observable effect according to the applied workload. The behavior of the SUT is indistinguishable from its behavior during the Golden Run, i.e. the SUT meets all the constraints concerning performance, transaction and system properties as specified in TPC-C [2] 3 and 5.

The SUT reports no errors of any kind.

All data remains consistent.

The SUT does not shut down.

**Degraded Performance (DP):** SUT behavior does not fall into fault effect class “No effect”.

The SUT reports no errors of any kind.

All data remains consistent.

The SUT does not shut down or crash.

The injected fault causes a delay in response time. I.e. the SUT does not meet the 90th percentile response time constraint defined in TPC-C [2] clause 5.7. The SUT must meet all other constraints concerning transaction and system properties as specified in TPC-C [2] clauses 3 and 5. The 90th percentile response time constraints for this fault effect class are summarized in table 5.2. This table is parametrized. The parameters  $\alpha_X$  must be chosen during benchmark implementation and documented in the full disclosure report. The following constraints must be fulfilled:

$$\forall X \in \{NO, P, OS, D\} : 5 < \alpha_X \quad (5.7)$$

$$30 < \alpha_{SL} \quad (5.8)$$

*NO*, *P*, *OS*, *D* and *SL* refer to the New-Order, Payment, Order-Status, Delivery and Stock-Level transactions respectively.

**Insufficient Performance (IP):** SUT behavior does not fall into fault effect class “Degraded Performance”.

The SUT reports no errors of any kind.

| Transaction Type  | 90th Percentile Response Time Constraint |
|---|--|
| New-Order   | $\alpha_{NO}$ sec.                       |
| Payment   | $\alpha_P$ sec.                          |
| Order-Status  | $\alpha_{OS}$ sec.                       |
| Delivery*   | $\alpha_D$ sec.                          |
| Stock-Level   | $\alpha_{SL}$ sec.                       |
| * The response time is for the terminal response (acknowledging that the transaction has been queued), not for the execution of the transaction itself. At least 90% of the transactions must complete within 120 seconds of their being queued (see clause 2.7.2.2). |  |

Table 5.2: Response Time Constraints for Fault Effect Class “Degraded Performance”

All data remains consistent.

The SUT does not shut down or crash.

The injected fault causes a delay in response time.

The system continues to respond to requests right up to the end of the measurement interval (i.e. the system does not hang).

**Detected Error (DE):** The SUT reports an error, either to the user, the operator or the error log.

All data remains consistent.

The SUT does not shut down or crash.

The SUT may or may not meet the response time or other constraints of TPC-C [2].

**Shutdown on Error (SE):** The SUT reports an error, either to the user, the operator or the error log.

All data remains consistent.

The SUT shuts down properly.

The SUT may or may not meet the response time or other constraints of TPC-C [2].

**Shutdown (SD):** The SUT does not report an error whatsoever.

All data remains consistent.

The SUT shuts down properly.

The SUT may or may not meet the response time or other constraints of TPC-C [2].

**System Crash/Hang (SC):** The SUT does not report an error whatsoever.

Data may or may not remain consistent.

The SUT does not shut down properly but crashes/hangs. I.e. after a certain point during the measurement interval, the SUT no longer responds to any requests.

The SUT may or may not meet the response time or other constraints of TPC-C [2].

**Bad Data (BD):** The SUT does not report an error whatsoever.

Data is inconsistent in the database or inconsistent data is delivered to the user.

The SUT does not shut down.

The SUT may or may not meet the response time or other constraints of TPC-C [2].

The SUT's behavior during each experiment must be assigned to exactly one fault effect class.

**Comment:** The term *system shutdown* is used to mean that the system shuts down properly, i.e. commits or rolls back transactions as required, closes open files, sends shutdown messages to connected clients, closes network sessions and after having done all this ceases to service any requests whatsoever until restarted.

The term *system crash* is used to mean that the system ceases to service any requests whatsoever until restarted *without* the performing all the actions for proper system shutdown.



## Clause 6

# SUT, Driver and Communications Definition

Clause 6 of TPC-C [2] is modified.

### 6.3 System Under Test (SUT) Definition

Clause 6.3 of TPC-C [2] is modified.

#### 6.3.5 The SUT consists of:

This clause is modified from clause 6.3.5 of TPC-C [2] in the following way: Any front-end systems are *not* considered to be part of the SUT. The rest of the clause remains unchanged.



## **Clause 7**

# **Pricing**

Clause 7 of TPC-C [2] is modified.

The measurements must be priced following the rules of clause 7, “Pricing”. The result will be price/tpmC.

These pricings must be weighted with the extended characteristics evaluating the effects of faults.

### **7.3 Maintenance**

The following statements are added to clause 7.3 of TPC-C [2]:

#### **7.3.5 Fault Recovery Pricing**

This clause is new.

The average cost for fault recovery and detection throughout the maintenance period must be priced. This cost must be calculated based on the MTBF of the components in the system, the cost of fault detection in the given system and the actual hardware/software and personnel cost to repair the component.

#### **7.3.6 Fault Effect Pricing**

This clause is new.

Average costs for fault effects based on MTBO of the failure modes and their costs.

### **7.4 Required Reporting**

The following statements are added to clause 7.4 of TPC-C [2]:

**7.4.3**

This clause is new.

The total cost  $X_j$  of each failure mode:

$$X_j = C_j + \sum_{i=0}^N ((c_i + d_{ji}) \cdot h_{ij}) \quad (7.1)$$

**7.4.4**

This clause is new.

The total failure cost  $X$  of the system:

$$X = \sum_{j=0}^M (X_j \cdot R_j) \quad (7.2)$$

## Clause 8

# Full Disclosure

Clause 8 of TPC-C [2] is modified.

### 8.1 Full Disclosure Report Requirements

Section 8.1.10 is added to 8.1 of TPC-C [2]:

#### 8.1.10 Clause 10 Faultload related items

This clause is new.

The following numerical quantities must be summarized near the beginning of the full disclosure report:

- The stationary availability  $A$ .

This clause is new.

The following information must be disclosed:

Failure Mode Table: completely filled as shown schematically in table 5.1.

$r_i$ : The fault rates of all faults used in the faultload (or MTBF instead).

$n_i$ : For all faults used in the faultload, the number of times this fault was applied during an experiment.

$n$ : The total number of experiments conducted.

$q_i$ : The repair rates of all faults used in the faultload (or MTTR instead).

$c_i$ : For all faults used in the faultload, the cost associated with repairing this fault.

Table of Detection Costs: completely filled as shown schematically in table 10.1.

$C_j$ : For all failure modes, the cost associated with this failure mode.

$R_j$ : The occurrence rates of all failure modes used (or MTBO instead).

$Q_j$ : The repair rates of all failure modes used (or MTTR instead).

$S_A, S_U$ : The sets used to calculate the stationary availability.

$X_j$ : The total cost of each failure mode.

$X$ : The total failure cost of the system.

## **Clause 9**

### **Audit**

The clause 9 does not currently apply to the dependability benchmark derived from TPC-C.



## Clause 10

# Faultload (FL)

This clause is new.

Experiments during which the SUT is submitted to a faultload must be executed, if the benchmark is to be published as a dependability benchmark.

This document covers hardware failures only.

A single fault is injected in one experiment.

### 10.1 Fault Mix

The faults injected in all experiments must meet the conditions detailed in the following paragraph.

Let  $r_i$  be the fault rate of fault  $f_i$ . Let  $\rho = \sum_i r_i$  be the sum of all the fault rates  $r_i$ . Let  $n$  be the total number of experiments we have the time to conduct. Then we can calculate the number of experiments  $n_i$  in which we inject fault  $f_i$  as

$$n_i = n \cdot \frac{r_i}{\rho}$$

**Comment:** In this way we make sure that numerical quantities for faults which occur often are calculated more exactly than those for faults which occur seldom.

### 10.2 Required Reporting

#### 10.2.1

This clause is new.

The fault rate  $r_i$  must be reported for each fault  $f_i$  in the faultload.

#### 10.2.2

This clause is new.

The repair rate  $q_i$  must be reported for each fault  $f_i$  in the faultload.

### 10.2.3

This clause is new.

The number of experiments  $n_i$  executed with fault  $f_i$  must be reported.

## 10.3 Fault Recovery Pricing

This clause is new.

### 10.3.1 Repair Cost

The cost associated with repairing fault  $f_i$  must be reported.

### 10.3.2 Table of Detection Costs

A table of detection costs must be reported. Table 10.1 shows an example.

The row-headings are the same failure modes which feature in table 5.1. The column-headings are the faults featuring in table 5.1. The values in the cells represent the cost  $d_{ji}$  needed to find out, that failure mode  $FM_j$  is indeed caused by fault  $f_i$ .

| Failure<br>Mode | Fault    |          |     |          |     |          |
|-----------------|----------|----------|-----|----------|-----|----------|
|                 | $f_0$    | $f_1$    | ... | $f_i$    | ... | $f_N$    |
| $FM_0$          | $d_{00}$ | $d_{01}$ | ... | $d_{0i}$ | ... | $d_{0N}$ |
| $FM_1$          | $d_{10}$ | $d_{11}$ | ... | $d_{1i}$ | ... | $d_{1N}$ |
| ...             | ...      | ...      | ... | ...      | ... | ...      |
| $FM_j$          | $d_{j0}$ | $d_{j1}$ | ... | $d_{ji}$ | ... | $d_{jN}$ |
| ...             | ...      | ...      | ... | ...      | ... | ...      |
| $FM_M$          | $d_{M0}$ | $d_{M1}$ | ... | $d_{Mi}$ | ... | $d_{MN}$ |

Table 10.1: Table of Detection Costs

# Appendix D

## Sample Faults

### D.1 Sample Disk Faults

Disk faults are differentiated to classes of disk faults:

- Whole disk: The whole disk does not respond to any command.
- Whole disk I/O: The whole disk responds neither to read nor to write requests.
- Head: One head of the disk is broken, causing read/write errors for the affected blocks.
- Track: One track is deleted, e.g. caused by a head crash, causing read/write errors for the affected blocks.
- Block: One block is defect, e.g. of manufacturing fault of the magnetic layer, causing read/write errors for the affected block.

Any disk part of the SUT may be subject to disk faults.

### D.2 Sample Network Faults

Network faults are differentiated between:

- receive loss: a packet is dropped at the receiving interface during the receive process with the (uniformly distributed) probability between 1% and 100%.
- send loss: a packet is dropped at the sending interface with the (uniformly distributed) probability between 1% and 100%.

**Comment:** Packets lost during receive create load on the net, whereas packets lost during send do not. Any network interface part of the SUT may be subject to network faults.

### D.3 Sample Processor Faults

Processor faults are differentiated according to registers. Each bit of a register may be the target as a fault of the following types:

- Bitflip: Bit changes spontaneously from 0 to 1 or vice versa (without being explicitly set by a program)
- Stuck at 0: Bit does not change its value any more, but always remains 0 (even when explicitly set to 1 by a program!)
- Stuck at 1: Bit does not change its value any more, but always remains 1 (even when explicitly set to 0 by a program!)

### D.4 Sample Memory Faults

Memory faults are differentiated to classes of memory faults:

- Bitflip: Bit changes spontaneously from 0 to 1 or vice versa (without being explicitly set by a program)
- Stuck at 0: Bit does not change its value any more, but always remains 0 (even when explicitly set to 1 by a program!)
- Stuck at 1: Bit does not change its value any more, but always remains 1 (even when explicitly set to 0 by a program!)

# Bibliography

- [1] DBench - Dependability Benchmarking (Project IST-2000-25425). Coordinator: Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique, Toulouse, France; Partners: Chalmers University of Technology, Göteborg, Sweden; Critical Software, Coimbra, Portugal; Faculdade de Ciências e Tecnologia da Universidade de Coimbra, Portugal; Friedrich-Alexander Universität, Erlangen-Nürnberg, Germany; Microsoft Research, Cambridge, UK; Universidad Politecnica de Valencia, Spain. URL: <http://www.laas.fr/DBench/>, 2001.
- [2] Transaction Processing Performance Council. TPC Benchmark [tm] C, Standard Specification, Revision 5.1, December, 2002. [www.tpc.org](http://www.tpc.org), 2002.