CITY UNIVERSITY LONDON

Software Design Diversity – from Conceptual Models to Practical Implementations

Dr Peter Popov
Centre for Software Reliability
City University London

ptp@csr.city.ac.uk
College Building, City University London EC1V 0HB
Tel: +44 207 040 8963 (direct)
+44 207 040 8420 (sec. CSR)

Software design diversity: Why

- The idea of redundancy (i.e. multiple software channels) for **increased reliability/availability** is not new:
  - has been known for a very long time and used actively in many application domains.
- simple redundancy **does not work** with software
  - software failures are deterministic: whenever a software fault is triggered a failure will result
  - software does not ware out
  - software channels work in parallel, but **must** be:
    - different by design (**design diversity**)
    - work on (slightly) different inputs/demands (**data diversity**)

CITY UNIVERSITY LONDON

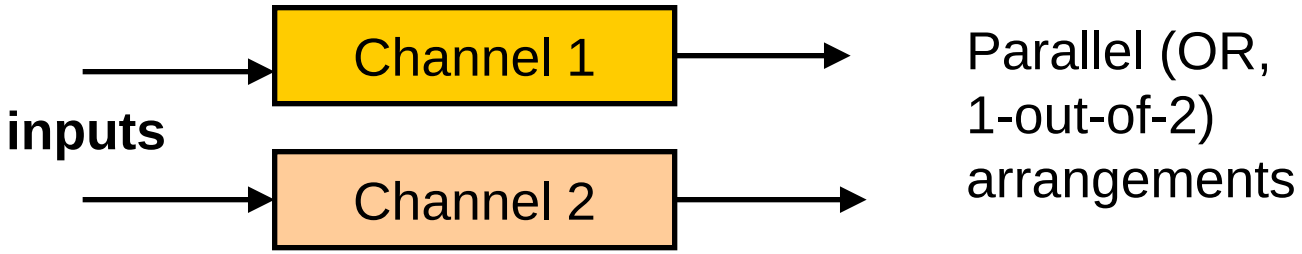
Software design diversity (2)

- **Surprisingly**, various homogeneous fail-over schemes dominate the market of FT 'enterprise' applications. **These are ineffective!**
- U.S.-Canada Power System Outage Task Force, Final Report on the August 14th (2003) Blackout in the United States and Canada
  - https://reports.energy.gov/BlackoutFinal-Web.pdf

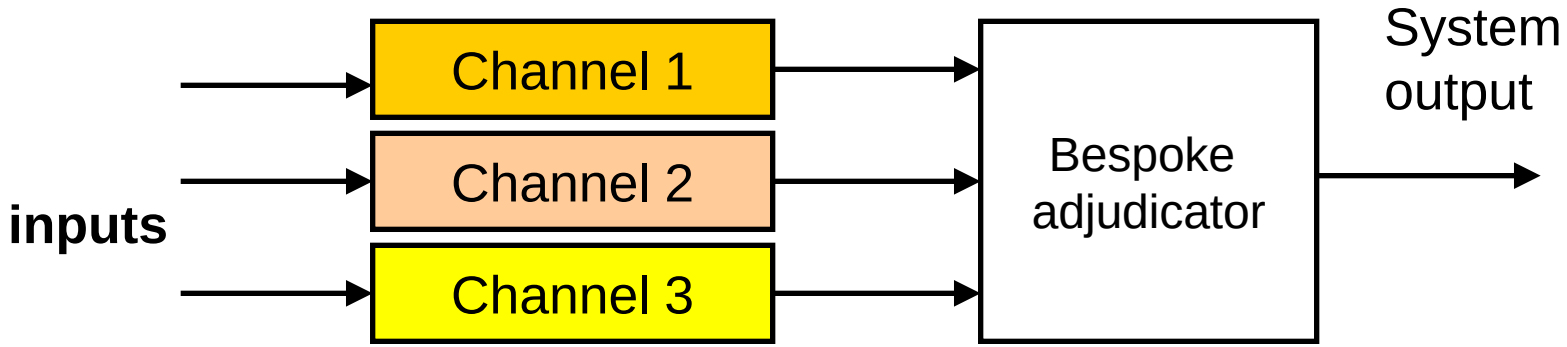
EMS Server Failures. FE's EMS system includes several server nodes that perform the higher functions of the EMS. Although any one of them can host all of the functions, FE's normal system configuration is to have a number of host subsets of the applications, with one server remaining in a "hot-standby" mode as a backup to the others should any fail. At 14:41 EDT, the primary server hosting the EMS alarm processing application failed, due either to the stalling of the alarm application, "queuing" to the remote EMS terminals, or some combination of the two. **Following preprogrammed instructions, the alarm system application and all other EMS software running on the first server automatically transferred ("failedover") onto the back-up server. However, because the alarm application moved intact onto the backup while still stalled and ineffective, the backup server failed 13 minutes later**, at 14:54 EDT. Accordingly, all of the EMS applications on these two servers stopped running.**(Part 2, p 32)**

Examples: diverse, modular redundancy

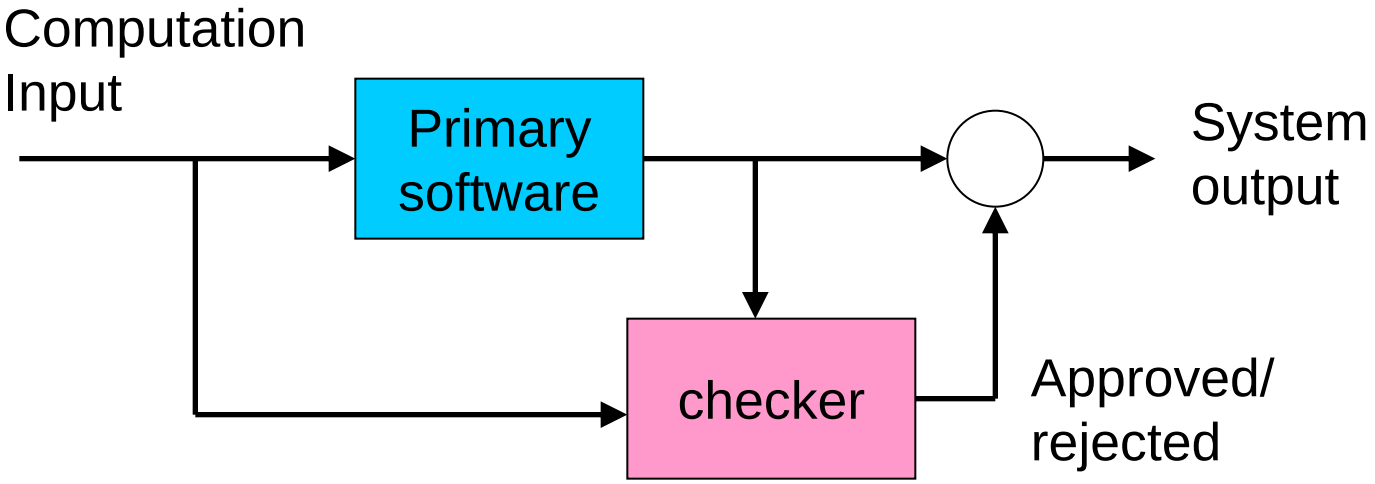
- "natural" 1-out-of-2 scheme (e.g. communication, alarm, protection)



- Voted system (e.g. control)



Examples: primary/checker systems



- Checker will usually be bespoke (possibly on OTS platform)
- If simpler than primary high quality is affordable
- Safety kernel idea can be implemented here

Achievement vs. Assessment

- Cost-benefit analysis is always needed:
  - design diversity is more expensive than non-diverse redundancy, or solutions without redundancy
    - especially in 80s, when the area was actively researched
  - what are the benefits of design diversity, how much one gains from diverse redundancy?
- Assessing the benefits is a problem much harder for (diverse) software than for hardware
- NVP 'implicitly' assumed ***independence of failures*** of the channels
  - *huge controversy*, very entertaining exchange in the IEEE Transaction on Software Engineering in mid 80s.

Is failure independence realistic?

- Knight and Leveson experiment (FTCS-15, 1985 and TSE, 1986)
  - 27 software versions developed to the same specification by students in two US Universities
  - tested on 1,000,000 test cases and the versions' reliability 'measured'
  - Coincident failures observed much more frequently than independence would suggest
    - i.e. refuted convincingly the hypothesis of statistical independence between the failures of the independently developed versions!
- Eckhardt & Lee model (TSE, 1985)
  - probabilistic model demonstrates why independently developed versions **will not fail independently**

Eckhardt and Lee model

- Model of software development
  - population of possible versions $\wp = \{\pi_1, \pi_2 ...\}$
  - probabilistic measure S(•), i.e. $S(\pi_i)$ is the probability that version π_i will be developed
- Demand space modelled probabilistically
  - $D = \{x_1, x_2 ...\}$ - demand space,
  - Q(•) probabilistic measure: the likelihood of different demands being chosen in operation.

$$\upsilon(\pi, x) = \begin{cases} 1, if\ \ program\, \pi\ \ fails\, on\, x; \\ 0, if\ \ program\, \pi\, does\, not\ \ fail\, on\, x. \end{cases}$$

Eckhardt and Lee model (2)

- The random variable $\upsilon(\Pi,X)$ represents the performance of a random program on a random demand: this is a model for the uncertainty both in software *development* and *usage*.

$$\theta(x) = \sum_{\wp} \upsilon(\pi, x).S(\pi) = \mathrm{E}_S\left(\upsilon(\Pi, x)\right)$$

is the probability that a randomly chosen program fails for a **particular** demand *x* ('difficulty' function).

- $\Theta \equiv \theta(X)$ is a **random variable**
  - upper case *X* represents a random demand, i.e. chosen in operation at random according to Q(•)

Eckhardt and Lee model (3)

$$P(\Pi_1 \, and \, \Pi_2 \, both \, fail \, on \, X) = P(\upsilon(\Pi_1, X)\upsilon(\Pi_2, X) = 1) =$$

$$\sum_{F} \sum_{\wp} \sum_{\wp} \upsilon(\pi_1, x)\upsilon(\pi_2, x) S(\pi_1) S(\pi_2) Q(x) =$$

$$\sum_{F} \left(\theta(x)\right)^2 Q(x) = \mathrm{E}\left(\Theta^2\right) = Var(\Theta) + \left(\mathrm{E}(\Theta)\right)^2 =$$

$$Var(\Theta) + \left(P(\Pi \, fails on \, X)\right)^2.$$

There is **no reason** to expect that independently developed software versions will fail independently on a randomly chosen demand, *X*, even though they fail conditionally independently on a given demand, *x*.

Littlewood and Miller model

- A generalisation of the EL model for the case of 'forced diversity'
  - the development teams are kept apart but also forced to use different methodologies, e.g. programming languages, different algorithms, etc.
- Model of forced diversity
  - probabilistic measures $S_A(\bullet)$ and $S_B(\bullet)$ for development methodologies, A and B.
    - a version (with a specific set of scores, $\upsilon(\pi,x)$) may be very likely with methodology A and very unlikely with methodology B
  - The model in every other aspect is identical to the EL model.

Littlewood and Miller model (2)

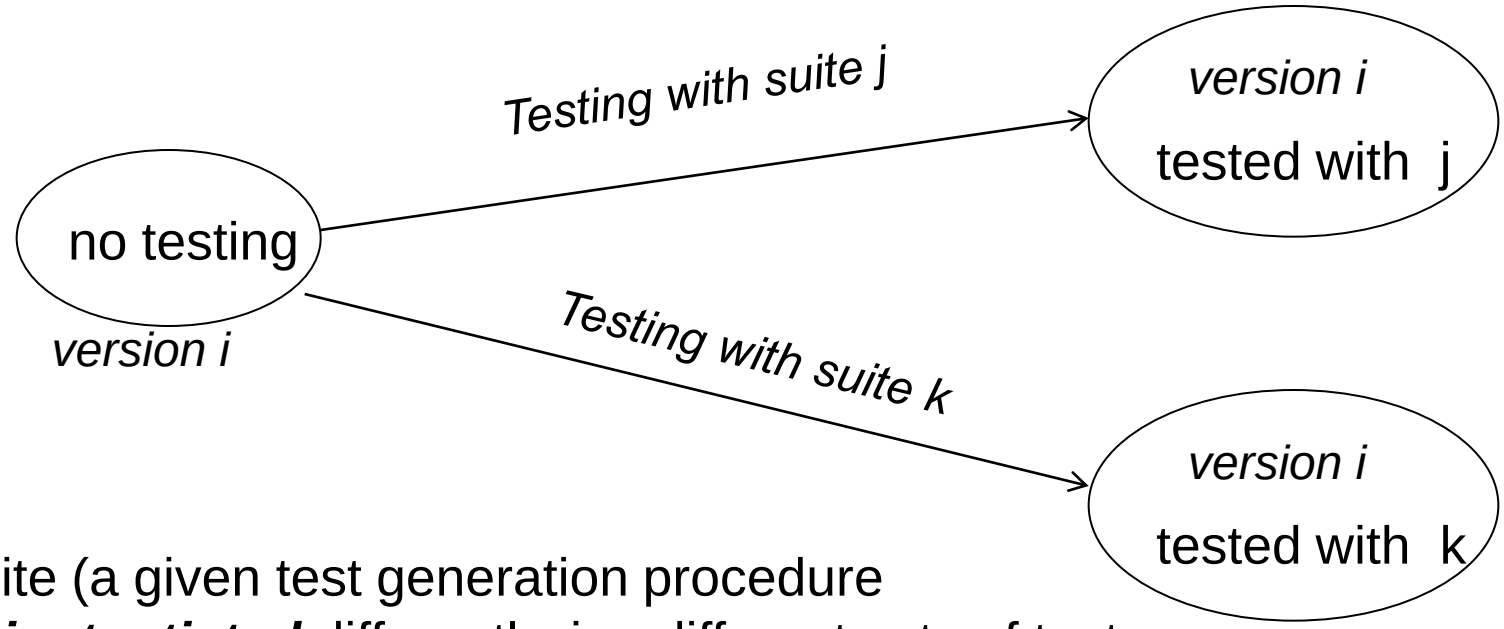
$$P(\Pi_A \text{ fails on } X, \Pi_B \text{ fails on } X) =$$
$$Cov(\Theta_A, \Theta_B) + P(\Pi_A \text{ fails on } X)P(\Pi_B \text{ fails on } X).$$

- Since **covariance can be negative**, then with forced diversity one may do even ***better than*** the unattainable ***independence*** under the EL model
- Littlewood & Miller in their TSE paper 1989 applied their model to Knight & Leveson's data and discovered **negative covariance.**
  - For them the two methodologies were represented by the programs developed by students from different universities.

Limitations of EL and LM models

- Eckhardt and Lee (EL) and Littlewood and Miller (LM) **models** deal with a **'snapshot'** of the population of versions
  - extended by allowing the versions to evolve through their being tested and fixing the detected faults
- These are models 'on average'
  - extended by looking at models of a particular pair of versions (models 'in particular').
  - Not covered here. The models are similar, but not identical.

A new model 'on average'



Testing:

- test suite (a given test generation procedure may be ***instantiated*** differently, i.e. different sets of test cases can be generated)
  - independently generated for each channel of the system;
  - the same test suite used;
- adjudication (oracle: perfect/imperfect, back-to-back)
- fault-removal (perfect/imperfect, new faults?)

Modelling the testing process

- $\Xi=\{t_1,t_2,\ldots\}$ with $M(\bullet)$, *i.e.* $M(t) = \mathrm{P}(T=t)$
- Extended score function:

$$\upsilon(\pi, x, t) = \begin{cases} 1, \text{ if } \pi \text{ tested with } t \text{ fails on } x, \\ 0, \text{ if } \pi \text{ tested with } t \text{ does not fail on } x. \end{cases}$$

$\upsilon(\pi, x, \varnothing)$ is the score of π on x before testing

Comparison of testing regimes

- Testing with oracles:
  - Detailed analysis with perfect oracles:
    - testing with oracles on **independently chosen testing suites**;
    - testing with oracles on the **same testing suite**;
  - Speculative analysis of oracle imperfection
- 'back-to-back' testing - lower and upper bounds identified under simplifying assumptions

In summary

- Performance of testing regimes (no account of the cost):
  - (**best** in terms of average system reliability achievable) independent testing with oracles;
  - (**worse**) testing with the same suite and oracles;
  - (**worst**) back-to-back testing.
- Accounting for the **cost may change this ordering**! A trade-off can be struck, which depends on cost of test suite generation and cost of testing.
- Counterintuitive observation:
  - forced diversity combined with testing with the same suite may lead to better system pfd than testing with independent suites (i.e. better result can be achieved more cheaply!)

Empirical Study with Database Servers

Fault-tolerance with off-the-shelf software becomes cheaper than with bespoke development, but what is the dependability gain, if any?

Empirical evidence is needed that the effort to build fault-tolerance with OTS is worthy

But what software to use?

- Toy examples? Open to criticism that findings are not applicable to complex software:
  - the gains may be very different between toy examples and 'real' complex software
  - difficulties of building FT solutions with diverse OTS s/w may be too high and the good idea is not practicable (Microsoft's concern)
- We avoid the first criticism by having decided to study complex OTS software such as RDBMS (SQL servers)

Overview of the study

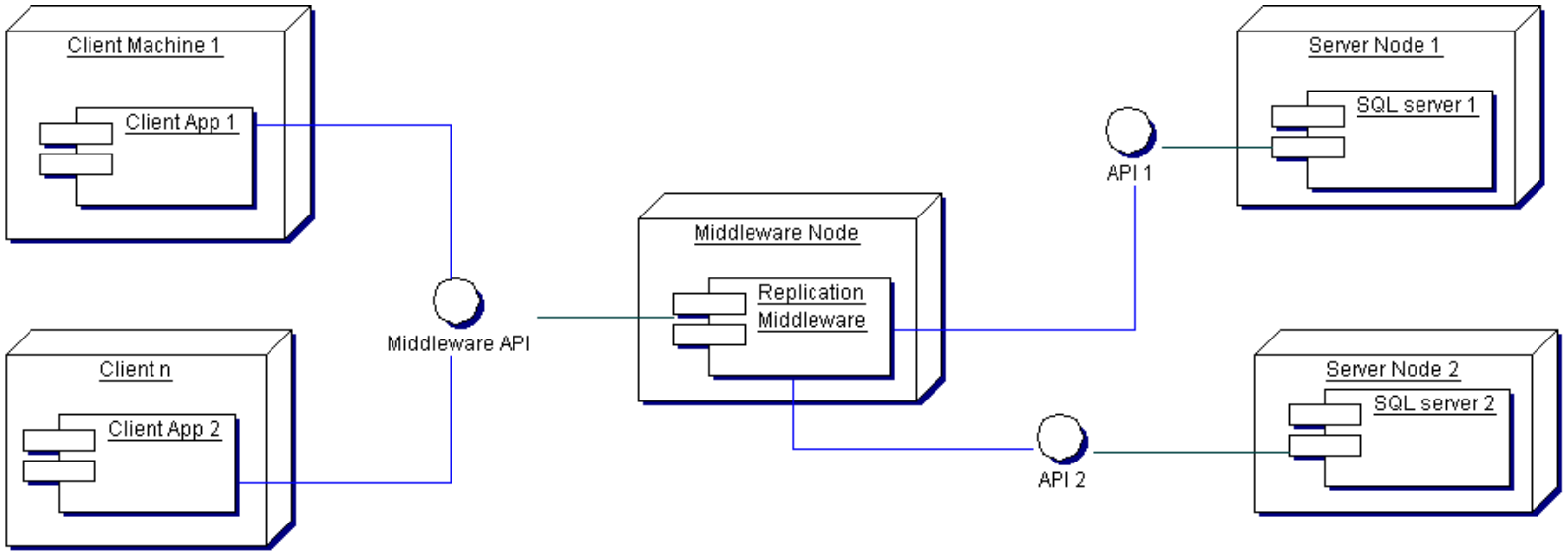
We have used SQL database servers - complex OTS products, with **many faults** (fixed) in each release. Standardisation exists (SQL-92 and SQL-99 standards), hence design diversity is realistic.

Difficulties

Differences in the syntax - manual translation was needed
- in parallel with the fault study a feasibility studies were undertaken as undergraduate student projects with automatic translation between the SQL dialects.

Many proprietary extensions in the servers, some impossible to 'translate' in another dialect (missing functionality)

Architecture for Fault-Tolerant database replication with Diverse SQL servers



- Effectiveness of the architecture in the end depends on the assumptions made about the failures
  - with database replication the assumption of 'fail-silent' failure (i.e. crashes) is **very common**
  - the studies allowed us to **validate this assumption** on the reported bugs.

Size of the 1st study

Servers included in the study:

- **Open source**:
  - PostgreSQL v. 7.0.0 (**PG**)
  - Interbase v. 6.0 (**IB**), now developed under the name Firebird
- **Commercial products (closed development)**:
  - Oracle v. 8.0.5 (**Oracle**)
  - MSSQL v. 7.0 (**MSSQL**)

181 known bug reports for all the servers together were collected.

1st study: IB faults

			IB	PG	OR	MS
Total Scripts			55	55	55	55
Script cannot be run (Functionality Missing)			n/a	23	20	16
Further Work			n/a	5	4	6
Total scripts run			55	27	31	33
No failure observed			8	26	31	31
Failure observed			47	1	0	2
Types of failures	Poor Performance		3	0	0	0
	Engine Crash		7	0	0	0
	Incorrect Result	Self-evident	4	0	0	1
		Non-self-evident	23	1	0	1
	Other	Self-evident	2	0	0	0
		Non-self-evident	8	0	0	0

1st study: PG, Oracle, MSSQL

			PG	IB	OR	MS	**OR**	IB	MS	PG	**MS**	IB	OR	PG
Total bug scripts			57	57	57	57	18	18	18	18	51	51	51	51
Bug script cannot be run (Functionality Missing)			n/a	32	27	24	n/a	13	13	12	n/a	36	32	31
Further Work			n/a	2	0	0	n/a	1	1	2	n/a	3	7	2
Total bug scripts run			57	23	30	33	18	4	4	4	51	12	12	18
No failure observed			5	23	30	31	4	4	4	3	12	11	12	12
Failure observed			52	0	0	**2**	14	0	0	**1**	39	**1**	0	**6**
Types of failures	**Poor Performance**		0	0	0	0	1	0	0	0	6	0	0	0
	Engine Crash		11	0	0	0	3	0	0	0	5	0	0	0
	Incorrect Result	**Self-evident**	14	0	0	**1**	3	0	0	0	10	0	0	**6**
		Non-self-evident	20	0	0	**1**	7	0	0	**1**	17	**1**	0	0
	Other	**Self-evident**	2	0	0	0	0	0	0	0	1	0	0	0
		Non-self-evident	5	0	0	0	0	0	0	0	0	0	0	0

CITY UNIVERSITY LONDON

1st study: 2 - Version combinations

Pairs of servers	Number of bug scripts run	Failure Observed (in at least one server)	One out of two servers fail		Both servers fail		
			Self-evident	Non -Self-evident	Non – Detect-able	Detectable	
						Self-evident	Non -Self-evident
IB + PG	62	43	17	25	1	0	0
IB + OR	62	29	8	21	0	0	0
IB + MS	69	35	11	21	2	1	0
PG + OR	64	30	13	16	0	0	1
PG + MS	66	46	18	21	1	6	0
OR + MS	71	14	7	7	0	0	0

Example: IB+PG Non-detectable bug

- Interbase Bug 223512(2)
  - both servers would drop Views using ***Drop Table*** statement.
    - violates of the SQL-92 standard, **Drop View** statement should be used.

1st study: Detectability of failures

Percentage of bugs causing **crash failure** varies between servers from 13% (MS SQL) to 21% (Oracle and PostgreSQL).

A non-diverse scheme would only detect the self-evident failures:

- crash failures,
- failures reported by the server itself (as exceptions) and poor performance failures.

For each of the four servers, less than 50% of bugs cause such failures.

With diverse pairs detectability is greatly improved:

- all the possible two-version fault-tolerant configurations detect the failures caused by **at least 94%** of the bugs used in the study.
- **None of the bugs caused a failure in more than two servers.**

Other issues:

- diagnosability (if different valid results received from the replicas which, if any, is giving us the correct answer?).
  - **Data diversity** (alternative but logically equivalent ways of formulating a query can be used to get from the same server multiple opinions and possibly diagnose the server 'changing its mind' – EDCC'06 reports on this aspect)
- Recovery (recovery blocks are very expensive with large DBs).

The second study

92 new bug reports were collected for the later releases of the **open-source DBMS** products:

- PostgreSQL 7.2 and Firebird 1.0 (the open-source descendant of Interbase 6.0.)

The closed-development DBMS products were excluded from the collection:

- Most of their bug reports lacked the bug scripts needed to trigger the faults.
- But the new bug scripts were still translated into the dialects of the closed-development ones, and were ran in the releases used in our first study (Oracle 8.0.5 and MSSQL 7.0).

The classification of faults and failures is the same as in the first study.

2nd Study - Analysis

Incorrect results are still the most frequent failures. Engine crashes are slightly more frequent than in the first study:

- but still no more than 22.2%.

The number of **non-self-evident failures is lower** than in the first study:

- 35% for PG 7.2 and 53% in FB.

The number of bugs causing coincident failures was again low:

- 5 coincident failures in total in the second study.

None of the bugs caused failures in more than two DBMSs.

Summary –the Bugs Studies

Out of the 273 bug scripts run in both studies:

- **very few bug** scripts affected two DBMS products;
- **none** affected more than two;
- only five of these bug scripts caused identical, **non-detectable failures** in two DBMS products:
  - of these five, one caused non-detectable failures on only a few among the demands affected.

The results of the *second study substantially confirmed* the general conclusions of the first study:

- the factors that make diversity useful do not seem to disappear as the DBMS products evolve (unclear if they have become more reliable)

Using successive releases of the *same* product for fault tolerance also appeared useful, although less so (not detailed here, but scrutinised in the IEEE TDSC'07 article)

Summary –the Bugs Studies (2)

There is strong evidence *against the fail-stop failure assumption* for DBMS products.

- The majority of bugs cause non-crash failures:
  - 64.5% (n.s-e.) vs. 17.1% in the first study; 65.5% vs. 19% in the second
- Even though these are bug reports and not failure reports, **this evidence goes against the common assumption that the majority of failures are engine crashes.**
- Users and designers of fault-tolerant solutions should, therefore, seek solutions to tolerate subtle and **non fail-silent failures**

It may be worthwhile for vendors to test their DBMS products using the known bug reports for other DBMS products.

- e.g. 4 MSSQL bugs were observed that had not been reported in the MSSQL service packs (previous to our observation period).

Similar observations have been reported recently by a MIT team (Barbara Liskov and her PhD student Ben Vandiver):

- More than 50% of the known bugs (DB2, MySQL, etc.) lead to non-self evident failures.

Details on the study

An article on both studies appears in the last issue of IEEE TDSC (October – December 2007) as ‘featured article’ with 900+ pages of supplement available online with **full detail** on bugs and the observations.

Commercial Exploitation

The empirical work was conducted by my PhD students.

- One conducted the fault study
- Another one conducted performance measurements and in the process developed an innovative replication protocol which we patented (European patent already granted, US is still pending)

With my younger colleagues we are trying to exploit commercially the benefits from software design diversity and build a highly reliable database storage using database replication with diverse databases.

Thank you

Questions