Using Covering Arrays to Construct a Benchmark For Concurrency Testing Tools

==

Using Combinatorial Test Design Techniques for Benchmark Construction

IBM Haifa Research Labs

Based on joint work of
Itai Segall, Eitan Farchi (IBM Haifa Research Labs)
Jeremy Bardbury, Kevin Jalbert and David Kelk
(University of Ontario Institute of Technology)

FITTEST

IBM

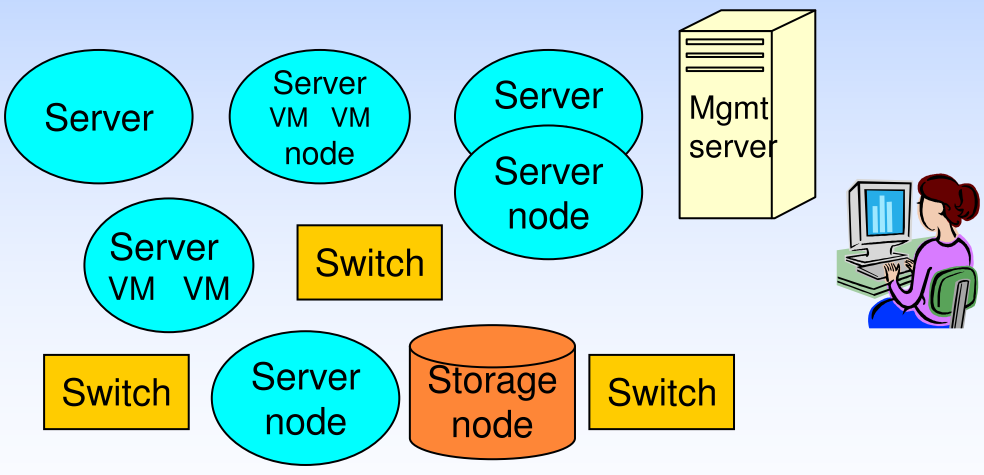
Background

Background – Concurrency Bug Detection Techniques

- Concurrency Testing
  - Coverage of the interleaving space in addition to coverage of the source code, inputs, etc.
- Static Analysis
  - Typically aimed at specific patterns or bugs
- Software Model Checking (exhaustive search)
  - Scalability issues
- No satisfactory benchmark exists
  - Effective
  - Small
  - Systematic

IBM Case Study System

- A distributed application for managing system resources in a networked environment.
  - Already described in IR10.2.
- Consists of:
  - a management server
    - application that communicates with multiple managed clients and with users of the management system.
  - Managed clients
    - physical or virtual resources distributed over a network.
- Managed resources include servers, virtual servers, storage devices, and network devices
- Used by IBM customers for managing IBM hardware and virtual devices, such as servers, Virtual Machines (VMs), switches and storage devices.
- The case study was performed on some new components of a version of this system which is still under development and has not yet been released for customer use.
  - Uses simulated environment



Background – Covering Arrays

- CA(k,v,t) is an array:
  - k columns
  - v possible values in each
  - All combinations of size t appear at least once

Background – Covering Arrays – cont.

- CA(4,2,2):

0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

Background – Covering Arrays – cont.

- CA(4,2,2):

0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

Background – Covering Arrays – cont.

- CA(4,2,2):

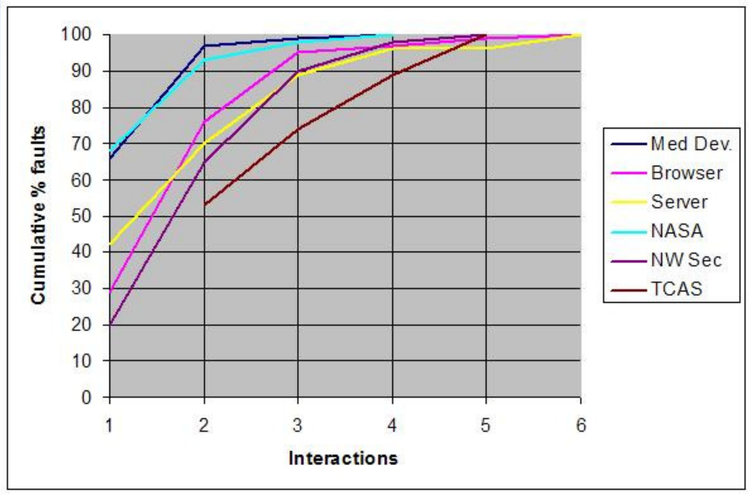
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

Background – Covering Arrays – cont.

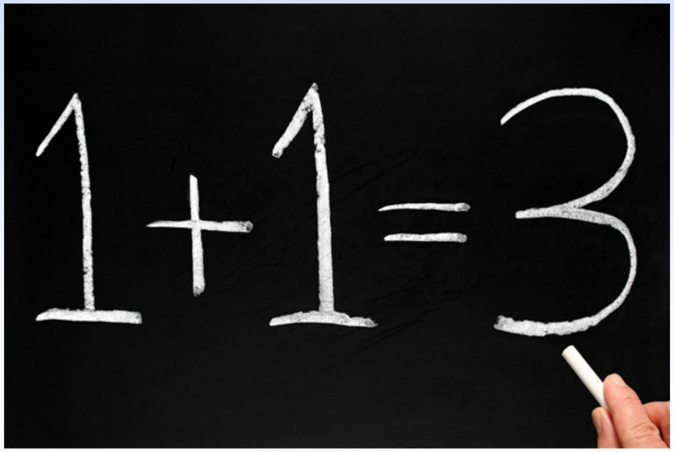
- Extensions to “pure” covering arrays:
  - Restrictions
  - Mixed-values
  - Mixed-strength
  - And many more…

Background – Combinatorial Test Design (CTD)

- CTD = Using covering arrays for testing
- Very well studied for 30 years
- Based on studies that show that software defects are caused by a small combination of attributes



Using Covering Arrays to Construct a Benchmark For Concurrency Testing Tools



Rationale

- Effective benchmark – differentiates between tools
  - For each pair of tools – have at least one task sample on which one outperforms the other
  - Typically, would want also another task sample on which the “other” outperforms the “one”
- (Intuition-only) claim: tool differentiation results from small combinations of factors

Rationale – cont.

- Exhaustive search tools vs. "noise-making" tools
  - Exhaustive tools choke on large programs and/or large # threads and/or large # critical regions
  - Noise-making tools will have difficulty finding certain bug patterns if the bugs are deep and/or rare
- "Noise-making" tools
  - Different heuristics are less/more effective based on combination of # threads and # critical regions

The proposed model

- Program size – # statements
- Program size – # critical regions
- Program size – % statements in critical regions
- Num threads
- Path error density
- Bug depth
- Bug pattern

Each combination here characterizes a (buggy) concurrent program

Pairwise plan for the model

	Program Size – # Statements	Program Size – # Critical Regions	Program Size – % Statements in Critical Regions	# Threads	Path Error Density	Bug Depth	Bug Pattern
1	Small	Small	Large	Small	Medium	High	TwoStageAccess
2	Medium	Small	Medium	Large	VeryLow	Medium	NonAtomicAssumedAtomic
3	Large	Medium	Small	Small	Low	Medium	BlockingCriticalSection
4	Medium	Large	Large	Medium	Low	Low	Interference
5	Small	Medium	Medium	VeryLarge	High	Low	OrphanedThread

… there are 44 such programs in our plan

… but you can't run a table
- or -
finding the needles in the haystack

- Our plan characterizes 44 combinations out of > 14,000 possible in the model
- No chance of finding actual programs that exactly match these

Mutants to the rescue…

- We therefore propose to:
  - Leverage existing benchmarks
  - Apply program mutation to generate new programs

Analyzing existing benchmarks

	Program Size – (sloc, statements)	Program Size – # Critical Regions	Program Size – % Statements in Critical Regions	Bug Pattern
account	Small(139,77)	Medium(7)	Small(11.6883)	NoLock
airlinestickets	Small(61,34)	Small(0)	Small(0)	Interference
allocationvector	Small(163,83)	Small(3)	Large(22.8916)	TwoStageAccess
boundedbuffer	Small(328,192)	Medium(5)	Large(20.3125)	NotifyInsteadOfNotifyAll
bubblesort	Small(236,84)	Small(3)	Medium(11.9048)	NonAtomicAssumedAtomic, OrphanedThread
bubblesort2	Small(98,43)	Small(1)	Medium(6.97674)	Initialization-Sleep

To be continued…

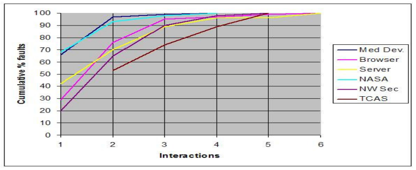
ConMAn

- 25 mutation operators, in 5 categories:
  - Modify parameters of concurrent methods
  - Modify the occurrence of concurrency method calls (re-moving, replacing, exchanging)
  - Modify concurrency keywords (addition and removal)
  - Switch concurrent objects
  - Modify critical regions (shift, expand, shrink, split)

Zooming out…

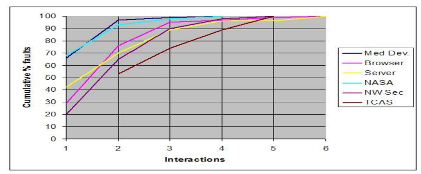
	Combinatorial Test Design	Combinatorial Benchmark Design

Zooming out…

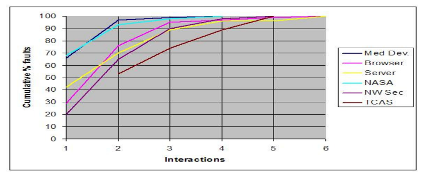
	Combinatorial Test Design	Combinatorial Benchmark Design
Rationale		(Intuitively) - differentiation results from small combinations

IBM

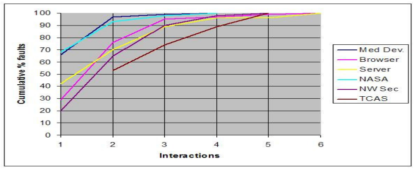
Zooming out…

	Combinatorial Test Design	Combinatorial Benchmark Design
Rationale		(Intuitively) - differentiation results from small combinations
Models	Describe points of variability in tests – potential causes of defects	Describe points of variability in task samples – potential causes of variability in tool effectiveness

Zooming out…

	Combinatorial Test Design	Combinatorial Benchmark Design
Rationale		(Intuitively) - differentiation results from small combinations
Models	Describe points of variability in tests – potential causes of defects	Describe points of variability in task samples – potential causes of variability in tool effectiveness
Analyzing existing artifacts	Tests commonly need to be abstracted to the level of the model	Samples in existing benchmarks need to be analyzed

Zooming out…

	Combinatorial Test Design	Combinatorial Benchmark Design
Rationale		(Intuitively) - differentiation results from small combinations
Models	Describe points of variability in tests – potential causes of defects	Describe points of variability in task samples – potential causes of variability in tool effectiveness
Analyzing existing artifacts	Tests commonly need to be abstracted to the level of the model	Samples in existing benchmarks need to be analyzed
Generation		Mutations

Thanks!

Questions ?