Random ... Architecture ... Predictions ...

Vlastimil Babka & Petr Tůma
DISTRIBUTED SYSTEMS RESEARCH GROUP
FACULTY OF MATHEMATICS AND PHYSICS
CHARLES UNIVERSITY IN PRAGUE
http://dsrg.mff.cuni.cz
Motivation ...

Validating software performance prediction

1. Get performance model
2. Get model annotations
3. Solve model

4. Instrument software
5. Measure software

6. Compare results

Mostly manual!
How well do we validate prediction?

**Few (often even just one) case studies**
- Very few shared case studies
- Realistic studies difficult to reproduce

**A lot of (invisible) manual adjustment**
- Studies done by experts with lots of time

**Strong motivation to present working results**
- Even without bad intent

Can the methods be intuitively trusted?
If we could just generate our software ...  
- We could generate the performance model  
- We could insert instrumentation easily  
- We could do this over and over  

The promise is  
- More thorough validation  
- Intuitive understanding of predictions  
- Perhaps even data mining over results?
Now for the pitfalls

The philosophical ones
- Is the generated software realistic?
- And is it realistic?
- Really?

The technical ones
- Algorithmic process of generation
- Where to get usable software fragments
- How to transform into performance model
It is not so bad after all ...

It does not have to actually **do** anything

All we need is realistic workload

- Especially where performance models simplify

What if we generate something unrealistic

- We can always exclude some results
- Prediction should work everywhere anyway ?
Simplifications in performance models

Operation durations
- Independent (invocations of the same operation)
- Independent (invocations of various operations)
- Exponentially distributed

Resource service strategies
- Ideal sharing or first-come first-served
- Only single resource used at a time
What we were thinking about ...

Design patterns
- At what level of granularity?
- Is there a really comprehensive list somewhere?

Software metrics
- Focused on human understanding
- Some are even difficult to automate
Algorithmic process of generation

Two types of modules

- Leaf modules
  ```
  void work (void) { ... };
  ```

- Interconnecting modules (seq, loop, branch)
  ```
  void work (void) { c1.work (); c2.work (); };
  ```

Top-to-bottom

- Pick a random module as root
- Recursively pick children if any interconnecting module lacks them
Example Control Flow and Architecture

Sequence

Branch

402.bzip2 fft 473.astar

Loop

444.namd

Sequence

Branch

429.mcf

Loop

count = 100

429.mcf

Branch

children = 3

402.bzip2 fft 473.astar 444.namd

429.mcf

bufsize=10K

Vlastimil Babka & Petr Tůma
Validating Model-Driven Performance Predictions On Random Software Systems
Generates tree architectures

- Sharing of components does not happen
- No communication between threads
- But some locking in there anyway

Difficult to adjust complexity

- Selection probability on each level configurable
- But still timing issues with nesting
  - Long loop over complex modules
  - Unbalanced branches
Usable software fragments

Realistic resource usage
- Picked existing benchmarks from the SPEC CPU suite
- Added some resource intensive modules

Turning benchmark into a module
- Shared static variables
  - Renamed during linking stage
  - Does not give thread isolation
- Invocation duration
Transforming into performance model

Two models supported right now

- Q-ImPrESS SAM
- SimQPN

Very different transformation issues

- Q-ImPrESS SAM almost one to one
- SimQPN more complicated
  - Queueing places for deployment nodes
  - Transitions for architectural control flow
SimQPN Transformation Details

- No formal meta-models
  - Transformation manually in Java
  - Components do not transform to the same QPN elements
SimQPN: Transformation of loops

- Tricky to avoid duplication of things inside loop body
  - The number of iteration has geometric distribution
Results: prediction with one client
Results: prediction with *multiple clients*
Results: \textit{norm} vs \textit{exp} service times
Results: $\text{exp} / \text{norm}$ service times ratio
Results: *isolated vs actual* timing durations

![Graph showing the ratio of measured to predicted service times against predicted mean CPU utilization in cores. The data points are scattered across the graph, with a notable concentration of points below the ratio of 1.0, indicating the comparison between isolated and actual timing durations.]
Results: impact of modules on precision

Graphs showing the relationship between CPU utilization and measured/predicted service times.
Results: Example of trend prediction

![Graph showing trend prediction](image)

- **Throughput [1/s]**
  - Y-axis range: 0.4 to 1.0
  - Data points at: 0.4, 0.6, 0.8, 1.0

- **Number of clients**
  - X-axis range: 2 to 14

- **Result type**
  - Simulated (exp)
  - Simulated (nor)
  - Measured

Vlastimil Babka & Petr Tůma
Validating Model-Driven Performance Predictions On Random Software Systems
Thank you ...