dBug: Systematic Evaluation of Distributed Systems

Jiří Šimša

Randy Bryant, Garth Gibson

PARALLEL DATA LABORATORY
Carnegie Mellon University
Concurrency Bugs Everywhere
Why Do Concurrency Bugs Exist?
Why Do Concurrency Bugs Exist?
Motivating Example Lessons

• Locking across RPC = bad idea
• Explosion of possible scenarios
• Corner case errors easy to miss

• Testing concurrent systems is hard:
  • Control / Enumerate possible scenarios
  • Tackle state space explosion
Need For Better Testing Methods

- Hardware performance
- Software complexity
- Formal specifications impractical
- New systems rarely written from scratch
- Common testing mechanism: stress testing
- Imprecise, falling behind
Outline

• Motivation
• dBug Design
• dBug Prototype
• Prototype Case Studies
• Ongoing & Future Work
• Conclusion
dBug Design

• Goal: Enable systematic enumeration of (all) possible execution scenarios of a test

• Repeated execution of the same test is guaranteed to explore different scenarios

• Light-weight model checking
  • Fixed initial state
  • User provided test as a specification
  • State space of the actual implementation explored
Motivating Example dBug-ed

Server 1 ... Server i ... Server n

Client ... Arbiter ... Client

Carnegie Mellon
Parallel Data Laboratory
http://www.pdl.cmu.edu/
dBug Design Decisions

• What events to control on and how?
• When to signal a request?
• How to (re)store a state of the system?
• How to explore the state space?
  • Parallel exploration
  • Exploration heuristics
  • State space reduction
Outline

• Motivation
• dBug Design
• dBug Prototype
• Prototype Case Studies
• Ongoing & Future Work
• Conclusion
Event Control Mechanism

Application

OS + Libraries

Application

dBug interposition

OS + Libraries
Compile-time Interposition

Source code annotation of:

- Creation of threads (processes)
- Destruction of threads (processes)
- Coordination primitives:
  - Thread synchronization
  - Remote procedure calls
  - “Your coordination primitive here”
Client-Server Architecture

Original Distributed System

Thread 1
dBug client

dBug

dBug server
dBug arbiter

Thread n
dBug client
When to Signal a Request?

• **Blind Mode:**
  • Uses a timeout
  • Pros: Easy to implement
  • Cons: Overhead, Imprecise

• **Informed Mode**
  • Uses application idle/progress hints
  • Pros: Fast, Accurate
  • Cons: Expert knowledge, Annotation
State Space Exploration
Outline

• Motivation
• dBug Design
• dBug Prototype
• **Prototype Case Studies**
• Ongoing & Future Work
• Conclusion
Fast Array of Wimpy Nodes

- Energy-efficient architecture
- FAWN-KV = distributed key-value storage
- `put()`/`get()` interface, strong consistency
- `get()` returns value of the last acked `put()`
Case Study 1: Multi-threading

- Obsolete data
- Up-to-date data

Log-structured writes
Need for clean-up

Rewrite Operation
- sequential scan
- atomic swap
Integrating FAWN-KV and dBug

• Creation and destruction of threads
  • 20 lines of annotations

• Acquiring and releasing locks
  • Compile-time interposition on pthread interface

• Test case:
  put(key,value1);
  if (fork() == 0) { rewrite(); }
  else { put(key,value2); get(key); }
Case Study Results

• Evaluated with the blind mode for ~24 hours
• Over 7000 possible scenarios
• Test always executed correctly

• Introduced and detected a data race bug
• The bug showed up in ~700 scenarios

• Two person weeks of work
Case Study 2: Including RPCs
Integrating FAWN-KV with dBug

• Creation and destruction of agents
  • 20 lines of annotations

• Issuing remote procedure calls
  • Modified Apache Thrift library (2 lines)

• Test case:
  put(key, value1);
  If (fork() == 0) { join(); }
  else { if (fork() == 0) put(key, value2); else get(key); }

http://www.pdl.cmu.edu/
Case Study Results

• Evaluated with blind mode for 45 minutes
• Total of 173 possible scenarios
• Found a bug

• The bug showed up in only 3 scenarios
• get(key) returns “not found”

• Two person weeks of work
Outline

• Motivation
• dBug Design
• dBug Prototype
• Prototype Case Studies
• Ongoing & Future Work
• Conclusion
dBug Evolution
dBug 2nd Generation

- Open source Autotools project
- dBug interposition as a shared library
- Precise and automatic detection of when to signal a request

- Educational use of dBug:
  - In use to evaluate student solutions for 15-213
  - Found bugs in the TA implementation
  - Available to students to test their solutions
Future Work

[Diagram showing different work support areas (Interface support, State Space, Case Studies, Event Injection) with past, present, and future technologies and tools such as Manual, Ad hoc, PVFS, FAWN-KV, POSIX threads, Parallel Exploration, RAIDTool, POSIX threads, FAWN-KV, Local I/O, Time Distortion, Local I/O, PVFS, and Network I/O.]
Outline

- Motivation
- dBug Design
- dBug Prototype
- Prototype Case Studies
- Ongoing & Future Work
- Conclusion
Related Work

- Verisoft [Godefroid98]
  - manual, exhaustive, multi-threaded, C and C++ sources
- MaceMC [Killian07]
  - automated, selective, distributed, Mace sources
- CHESS [Musuvathi08]
  - automated, selective, multi-threaded, Windows binaries
- MoDist [Yang09]
  - automated, selective, distributed, Windows binaries
Conclusion

- Systematic and automatic evaluation of distributed system test cases
- Open source implementation of dBug
- Experiments with:
  - Parallel Virtual File System (C)
  - FAWN-based key value storage (C++)
  - CMU student class projects (C and C++)
  - RAIDTool (Java)
- Finding real bugs
References