Focus Replay Debugging Effort on the Control Plane

Gautam Altekar and Ion Stoica
University of California, Berkeley

Debugging Software Is Hard

Debugging datacenter software is *really* hard

Datacenter software?

- Large-scale, data-intensive, distributed apps

Hard?

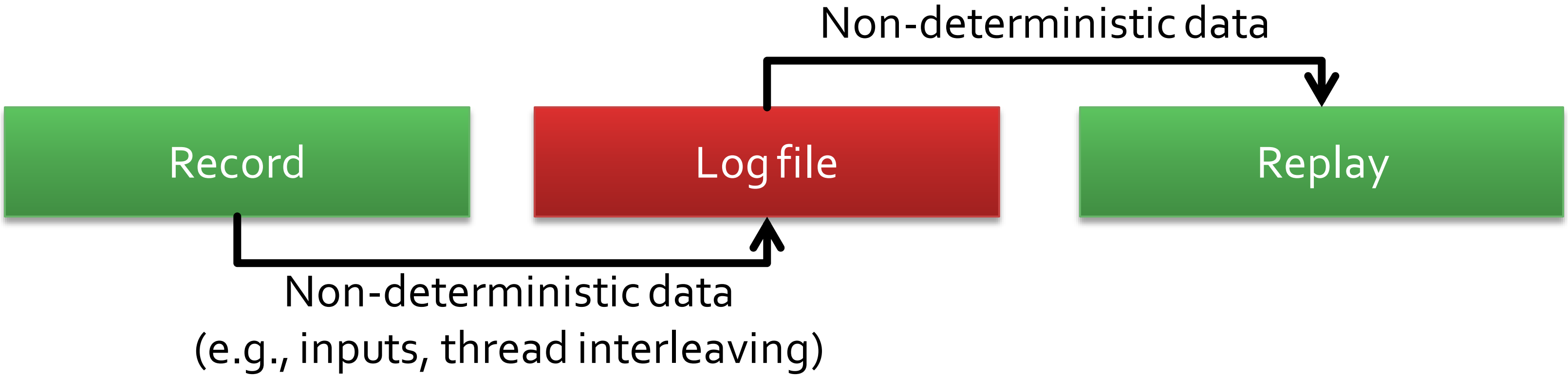
Non-determinism

- Can't reproduce failures
- Can't cyclically debug

How can we reproduce non-deterministic failures in datacenter software?

Deterministic Replay Systems

Generate replica of original run, hence failures



Non-deterministic data

Record

Log file

Replay

Non-deterministic data
(e.g., inputs, thread interleaving)

Why deterministic replay?

- Model checking, testing, verification
  - Goal: find errors pre-production
  - Can’t catch all errors
  - Can’t reproduce production failures

Requirements for Datacenter Replay

- Always-on production use
  - < 5% slowdown
  - Log no more than traditional console logs (100 Kbps)
- High fidelity replay
  - Reproduce the most difficult of non-deterministic bugs

Related Work

None suitable for the datacenter

	Always-on operation?	High fidelity replay?
FDR, Capo, CoreDet	**No**	Yes
VMWare, PRES, ReSpec	Yes	**No**
ODR, ESD, SherLog	Yes	**No**
R2	Yes	**No**

Goal

Build a Data Center Replay System

Target

- Record efficiently ~20% overhead, 100 KBps
- High replay fidelity
  - Replays difficult bugs

Design for

- Large-scale, data-intensive, distributed apps

- Linux/x86

Outline

- ✓ Overview
- ➢ **Approach**
- ➢ Testing the Hypothesis
- ➢ Preliminary Results
- ➢ Ongoing Work

Control Plane Determinism: Intuition

For debugging, not necessary to produce identical run

Often suffices to produce *any* run that has same control-plane behavior

The Control Plane?

Datacenter apps have two components

1. Control-plane code

Manages the data

Complicated, Low traffic

- ➢Distributed data placement
- ➢Replica consistency

2. Data-plane code

Processes the data

Simple, High traffic

- ➢Checksum verification
- ➢String matching

Our Hypothesis

Relax guarantees to control-plane determinism

Meet all requirements for a practical datacenter replay system

Outline

- ✓ Overview
- Approach
- **Testing the Hypothesis**
- Preliminary Results
- Ongoing Work

Testing Criteria

Experimentally show the control plane has:

1. Higher bug rates, *by far*
   - Most bugs must stem from control plane code
   - Implies high fidelity replay
2. Lower data rates, *by far*
   - Consumes and generates very little I/O
   - Implies low overhead recording

Test Results - Preview

Control Plane

99%

Bug Rate

1%

Data Rate

Data Plane

1%

Bug Rate

99%

Data Rate

Evidence support the hypothesis

Outline

- ✓ Overview
- Hypothesis
- **Testing the Hypothesis**
  - **How?**
- Preliminary Results
- Ongoing Work

Challenge: Classification

- To make statements about planes, we must first identify them
- Goal: Classify code as control and data plane code
  - Hard: tied to program semantics
- Obvious approach: Manually identify plane code
  - Error prone and unreliable

Approach: Semi-Automated Classification

1. Manually identify user-data files
   - User data? E.g., file uploaded to HDFS
2. Automatically identify static instructions tainted by user data
   - Taint-flow analysis
3. Instructions tainted by user data are in data plane; others are in control plane

Taint Flow Analysis

- Instruction-level
  - Works with apps written in arbitrary languages
- Dynamic
  - Easier to get accurate results (e.g., in the presence of dynamically generated code)
- Distributed
  - Avoids need to identify user-data entry points for each component

Classifier Limitations

- It’s imprecise
  - We may have misidentified user data (unlikely)
  - We don’t propagate taint across tainted-pointer dereferences (to avoid false positives)
- It’s incomplete
  - Dynamic analysis often has low code coverage
  - Results do not generalize to arbitrary executions

Outline

- ✓ Overview
- ➢ Hypothesis
- ➢ Testing the Hypothesis
- ➢ **Evaluation**
- ➢ Ongoing Work

Evaluation Setup

- Distributed applications
  - Hypertable: Key-value store
  - KFS/CloudStore: Filesystem
  - OpenSSH (scp): Secure file transfer
- Configuration
  - 1 client, 1 of each system node
  - 10 GB user-data file
  - Kept simple to ease understanding

Evaluation Metrics

- Bug rates
  - Indirect: code size (static x86 instructions executed)
  - Direct: Bug-report count (Bugzilla)
- Data rates
  - Fraction of total I/O

Outline

- ✓ Overview
- Hypothesis
- Testing the Hypothesis
- **Evaluation**
  - **OpenSSH**
- Ongoing Work

Bug Rates: Code Size

OpenSSH: Executed Static Instructions

	Control (%)	Data (%)	Total (K)
Agent	100	0	11
Server	97.8	2.2	103
Client (scp)	98.9	1.1	69
Average	**98.9**	**1.1**	**61**

Even components that touch user-data are almost exclusively control plane

Bug Rates: Report Count

OpenSSH: Bugzilla Report Count

	Control (%)	Data (%)	Total
Agent	100	0	2
Server	100	0	215
Client (scp)	99	1	153
Average	**99.7**	**0.3**	**123**

Control plane is the most error-prone, even in components that touch user-data

Control Plane is More Bug-Prone. Why?

(1) Control plane executes many functions to perform its core tasks

OpenSSH: # of functions hosting top 90% of dynamic instructions

	Control	Data
Agent	13	0
Server	100	1
Client (scp)	27	1
Average	**47**	**1**

Most active data plane functions: aes_encrypt() and aes_decrypt()

Control Plane is More Bug-Prone. Why?

(2) Control plane relies heavily of custom code

OpenSSH: % of Dynamic Instructions Issued from Libraries

	Control (%)	Data (%)
Agent	82.7	0
Server	93.6	99.6
Client (scp)	96.2	100
Average	**90.8**	**99.8**

Data plane often relies on well-tested libraries (e.g., libc, libcrypto, etc.)

Data Rates: A Closer Look

What should I say here?

	Control (%)	Data (%)	Total (GB)
Agent	100	0	0.001
Server	0.8	99.2	20.2
Client (scp)	0.6	99.4	20.2

Ongoing Work

- How well do results generalize?
  - To other code paths
  - To other applications
- How do we achieve control plane determinism?
  - Should we just ignore the data plane?
  - Should we use inference techniques?

Conclusion

What have we argued?

Control-plane determinism enables record-efficient, high-fidelity datacenter replay

What's next?

More application data points

Questions?