Holistic System Design for Deterministic Replay

Dongyoon Lee
Computer Science Department
DVR: Never Miss a Show

DVR - Control recorded TV

Record
Play
Pause
Fast-Forward
Rewind
Stop
Deterministic Replay

- Record an execution
- Reproduce the same execution

Tradional Debugging

\[ p = \text{NULL} \]

Is \( p == \text{NULL} \)?

watchpoint

Replay-based Debugging

\[ \leftarrow \text{Step backward} \]
Deterministic Replay

- Record an execution
- Reproduce the same execution
Deterministic Replay

- Record an execution
- Reproduce the same execution

OFFLINE Program Analysis

Audit
Forensics
Problem Diagnosis

Heavyweight

Less overhead
Less distortion

Deterministic Replay

Forensics, Auditing, ...
Deterministic Replay

- Record an execution
- Reproduce the same execution

Offloaded Security Checks

ONLINE Program Analysis

Smartphone

Server-side Replica

Resource Constraints

Security Checks

RECORD

LOG

REPLAY

Online Replay
Deterministic Replay

- Record an execution
- Reproduce the same execution

Fault Tolerance

Always-on bank service

client

server

replica

Deterministic Replay

Online Replay
Deterministic Replay

- **Record** an execution
- **Reproduce** the same execution

Deterministic Replay

- Offloaded Security Checks
- Forensics, Auditing, ...
- Architecture simulation
- Fault Tolerance
- Time-travel Debugging

Approximately

< DVR >

**Uniprocessors**
- At low overhead (~5%)
- Commercial products

**Multiprocessors**

2004

SOLVED!
Past Approach for Multiprocessor Replay

Thread 1  Thread 2  Thread 3

Checkpoint  Initial State

Log data from the external world
(program input)

Log shared-memory dependencies

Write
Write
Read

10-100x
Log dependencies *imprecisely*, but still provide useful *replay guarantee*.
Replay Guarantee

Value Determinism
- Same sequence of instructions, reading/writing same value

Schedule Determinism
- Same thread interleaving

External Determinism
- Same output and final state

Debugging

Offloaded analysis
Security Checks
smartphone
replica

Fault tolerance
client
server
replica

Past solutions log precise shared-memory dependencies

Past solutions log precise shared-memory dependencies
Roadmap

What is Replay?

On-going Work

Doubleplay

Respec OS

Online Replay (replication)

Offloaded analysis

Fault tolerance

Debugging

Offloaded analysis

Security Checks

fault tolerance

client

server

replica

replica

CRASH!
Overview of Respec

Checkpoint  Initial State

Log data from the external world

Log the subset of shared-memory dependencies enough for **External Determinism**
Recording for Data-Race-Free Programs

Checkpoint  Initial State

Log data from the external world

Data-Race-Free
- Different threads
- Synchronized operations
- Write operation

Thread 1

Lock
X = 1
Unlock

Thread 2

X = 0

Thread 3

Lock
X = 2
Unlock

Thread 1

Thread 2

Thread 3
Online Replay with Speculation

- Record and replay program input
- Record and replay the order of synchronizations
  - Speculate data-race-free programs
What if a program is **NOT** race free?

**Problem**
- Mis-speculation
- Data race detector?

**Insight:** *External Determinism* is sufficient
- Guarantee the same *output* and *final state*

**Solution**
- Detect mis-speculation when the replay is not *externally deterministic*
• Compare system call arguments
• Guarantee external determinism

Check #1 – System Output

Recorded Process

Replayed Process

Check $O' == O$?
Inconsequential Data Races

- Not all races cause external determinism checks to fail

```
x=1
x!=0?
```

```
mul<-threaded
fork
```

```
T1
T2
Start A
x=1
x!=0?
SysWrite(x)
```

```
T1'
T2'
Start A'
x=1
x!=0?
SysWrite(x)
```

Recorded Process

Replayed Process

Success
Divergence due to Data Races

1) Need to rollback to the beginning
2) Need to buffer system output till the end
Check #2 – Program State

- Create a **checkpoint** periodically
- Compare **memory** and **register** states

Recorded Process

Replayed Process
• Optimistically re-run the failed epoch
• On repeated failure, record and replay only one thread at a time
Implementation and Evaluation

Implementation
- Modified Linux kernel and glibc library

Test Environment
- 2 GHz 8 core Xeon processor with 3 GB of RAM
- Run 1~4 worker threads (excluding control threads)

Benchmarks
- PARSEC suite
  - blackscholes, bodytrack, fluidanimate, swaptions, streamcluster
- SPLASH-2 suite
  - ocean, raytrace, volrend, water-nsq, fft, and radix
- Desktop/Server applications
  - pbzip2, pfscan, aget, and Apache
Record and Replay Performance

- 18% for 2 threads, 55% for 4 threads
Roadmap

Doubleplay

What is Replay?

Respec

OS

Rosa

HW

Future Work

Online replay ➔ Offline replay
Recording Thread Schedule for Offline Replay

Logging thread schedule is sufficient for deterministic replay

Need to wait until the online replay finishes (4x slow)
DoublePlay: Uniparallel Execution

- Provide the **benefits** (guarantees) of uniprocessor executions
- Provide the **performance** of multiprocessor executions
Roadmap

What is Replay?

Respec OS

Future Work

Software solution
~30% slowdown

Hardware support
<1% slowdown
ROSAX (Replay via Offline Symbolic Analysis)

Thread 1  Thread 2  Thread 3

Checkpoint Initial State

Load-based Hardware Recorder*

Read

Write

Satisfiability-Modulo-Theory (SMT) solver
reconstructs thread interleaving **offline**
Load-based Logging Architecture

Insight

• Recording initial register state and values of loads is sufficient for deterministic replay

Optimization

• Record a load only if it is the first access to a memory location

ROSA: Processor support

• Recording data fetched on a cache miss captures first loads
Cache Miss Logging for Single-threaded Programs

**< Recording >**

- **Checkpoint**
  - R1 = Load A
  - R2 = Load B
  - R3 = Load A
  - R3 = R3 + 1
  - Execution Time
  - Cache Miss

- Log file
  - R1 = R2 = R3 = 0
  - <cnt1, 1>
  - <cnt2, 2>

**< Replay >**

- **Checkpoint**
  - R1 = Load A
  - R2 = Load B
  - R3 = Load A
  - R3 = R3 + 1

- Reg.
  - Val.
  - R1
    - 1
  - R2
    - 2
  - R3
    - 2

- Mem.
  - Val.
  - A
    - -
  - B
    - -
  - ....
    - ...

Mem.
• Logging cache misses for each thread is sufficient for replaying that thread
Shared Memory Dependency

**Final State**: A, B, C

*SMT Solver* for finding shared memory dependencies
Offline Symbolic Analysis

Step 1) Encode Ordering Constraints

- Coherence Constraints
- Program Order Constraints

Final State

Coherence Constraints
(X1 < X2) AND
(X2 < X4 OR X5 < X2) AND
...

Program Order Constraints
(Y1 < X1 < X2 < Y2) AND
(X3 < X4 < X5 < Y3) AND
...

Step 2) Find a valid casual order

- Satisfiability Module Theory (SMT) Solver
- Yices [Dutertre and Moura CAV’06]
Encoding Coherence Constraints

Coherence Constraints
( M→old == M→prev→new)

X1: X1 < X3  AND
X2: (X3 < X2 < X4 OR X5 < X2) AND

Address ⏎ Old value
                  ⏎ New value
Encoding Program Order Constraints

Coherence Constraints
(\(M \rightarrow \text{old} = M \rightarrow \text{prev} \rightarrow \text{new}\))

X1: \(X1 < X3\) AND
X2: \((X3 < X2 < X4 \text{ OR } X5 < X2)\) AND
Y1: \(Y1 < Y2\) AND
Y2: \(Y1 < Y2 < Y3\) AND

Program Order Constraints
(under Sequential Consistency)

P1: \(Y1 < X1 < X2 < Y2\) AND
P2: \(X3 < X4 < X5 < Y3\) AND
Replay Guarantee of ROSA

**Schedule Determinism**
- Same thread *interleaving*

**Value Determinism**
- Same *sequence* of instructions, reading/writing same *value*

**External Determinism**
- Same *output* and *final state*
Evaluation

Simulation Framework

- **Simics**: simulate multi-processor execution (2, 4, 8, 16 cores)
- Fast-forward up to known synchronization points
- Simulated for 500 million instructions

Benchmarks

- SPLASH2: barnes, fmm, ocean
- PARSEC2.0: blackscholes, bodytrack, x264
- SPEComp: wupwise, swim
- Servers: apache, mysql

Offline Symbolic Analysis

- **Yices** SMT solver [Dutertre and Moura CAV’06]
Cache Miss Log Size

- On average, 192 MB/sec (8 threads)
- Performance overhead: <1% slowdown in IPC
On average, 260 seconds/sec (8 threads)
One time cost before replay
Roadmap

What is Replay?

Respec OS

Rosa HW

On-going Work

“Make easy-to-use and reliable parallel systems”
Support for Next-Generation Parallel Systems

Distributed Systems (Cloud Computing)

Large-scale Migration Network

Mobile Systems

Event-driven Programs Resource Constraints

Problem Diagnosis Debugging Testing
Why Event-Driven Programming Model?

Need to process **asynchronous input** from a rich set of sources

Slides extracted from “Hsiao et al’s PLDI 2014 presentation”
Events and Threads in Android

Looper

Regular Threads

Event Queue

send()

signal(m)

wr(x)

wait(m)

rd(x)

onClick() {
...
}

onServiceConnected() {
...
}
Non-determinism: Multi-thread Data Race

Looper Thread

- onClick()
- onServiceConnected()

Regular Threads

- send(m)
- signal(m)
- wait(m)
- rd(x)
- wr(x)

Causal order: happens-before (→) defined by synchronization operations

Conflict: Read-Write or Write-Write data accesses to same location

Race (↔): Conflicts that are not causally ordered

Slides extracted from “Hsiao et al’s PLDI 2014 presentation”
Looper Thread

onReceive() {
    *p;
}

onDestroy() {
    p = null;
}

onClick() {
    send( );
}

Regular Threads

NullPointerException!

Non-determinism:
  Single-thread
  Data Race

Slides extracted from “Hsiao et al’s PLDI 2014 presentation”
Conclusion

• How can we help programmers to write reliable and secure concurrent systems?

• Deterministic multiprocessor replay with external determinism
  • Log non-deterministic events during recording (partially)
  • Reproduce them during replay (guaranteeing the same output)

• On-going research
  • Concurrency analysis for concurrent mobile programs
    – Static/dynamic data race detection
    – Static/dynamic taint analysis
    – Offloaded runtime checks via replay