Finding and Reproducing Heisenbugs in Concurrent Programs

Madanlal Musuvathi Shaz Qadeer Thomas Ball Gerard Basler

Microsoft Research

OSDI '08

Presentation given by Wang Yuanxuan

Outline

Outline

Heisenbugs

Definition

Subtle interactions among threads and the timing of asynchronous events can result in concurrency errors that a hard to find, reproduce and debug. Stories are legend of so-called "Heisenbugs" that occasionally surface in systems that have otherwise been running reliably for months.

CHESS

Definition

A tool for systematic and deterministic testing of concurrent programs.

Features

CHESS takes complete control over the scheduling of threads and asynchronous events.

- Capability to reproduce the erroneous thread interleaving.
- Systematic enumeration techniques to force every run of the program along a different threading interleaving.

CHESS

Definition

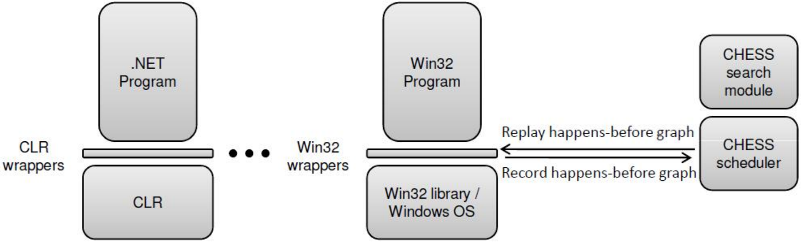
A tool for systematic and deterministic testing of concurrent programs.

Challenges

Challenges to build a systematic testing tool for concurrent programs

- Avoid perturbing the system under test.
- Accomplish the nontrivial task of capturing and exploring all interleaving nondeterminism.
- Explore the space of thread interleavings intelligently.

CHESS Architecture



Outline

Example

Example

How CHESS was used to reproduce a Heisenbug in CCR, a .NET library for efficient asynchronous concurrent programming.

The Bug

The entire test run consists of many smaller unit concurrency tests. The failing test(which did not terminate) previously had not failed for many months.

Steps

- Changed the harness so it ran the test just once.

Steps

- Changed the harness so it ran the test just once.
- Ran the test under CHESS

Steps

- Changed the harness so it ran the test just once.
- Ran the test under CHESS
- In just over 20secs, CHESS reported a deadlock after exploring 6737 different thread interleavings.

Steps

- Changed the harness so it ran the test just once.
- Ran the test under CHESS
- In just over 20secs, CHESS reported a deadlock after exploring 6737 different thread interleavings.
- Let CHESS reproduce the last deadlock scenario.

Steps

- Changed the harness so it ran the test just once.
- Ran the test under CHESS
- In just over 20secs, CHESS reported a deadlock after exploring 6737 different thread interleavings.
- Let CHESS reproduce the last deadlock scenario.
- Ran CHESS on the offending schedule under the control of a standard debugger.

Steps

- Changed the harness so it ran the test just once.
- Ran the test under CHESS
- In just over 20secs, CHESS reported a deadlock after exploring 6737 different thread interleavings.
- Let CHESS reproduce the last deadlock scenario.
- Ran CHESS on the offending schedule under the control of a standard debugger.
- The source of the bug was identified.

Outline

Primary Goal

- Capture all the nondeterminism during a program execution.
- Expose these nondeterministic choices to a search engine

Handling input nondeterminism

- Shift the onus of generating deterministic inputs to the user.
- Consider an elaborate log and replay mechanism unnecessary.
- Log and replay input values that are not easily controlled by the user.

A Wrapper Library

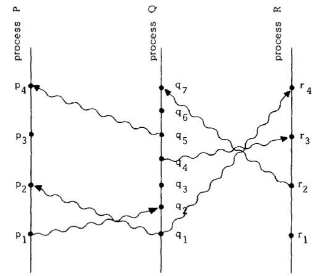
- The scheduler redirects calls to concurrency primitives.
- By including complex primitives as part of the program, the scheduler only needs to understand the simpler primitives.

The happens-before graph

Definition

The graph capturing the relative execution order of the threads in a concurrent execution.

Example



The happens-before graph

Definition

The graph capturing the relative execution order of the threads in a concurrent execution.

Benefits

- Provide a common framework for reasoning about all the different synchronization primitives used by a program.
- Abstract the timing of instructions in the execution.

The happens-before graph

Definition

The graph capturing the relative execution order of the threads in a concurrent execution.

Node

Each node is annotated with a triple
(task, synchronization variable, operation)

The happens-before graph

Definition

The graph capturing the relative execution order of the threads in a concurrent execution.

Tasks

- Threading executing an instruction
- Threadpool work items
- asynchronous callbacks
- timer callbacks

The happens-before graph

Definition

The graph capturing the relative execution order of the threads in a concurrent execution.

Operations

CHESS only needs to understand

- isWrite. if true, two sets of edges are created)
- isRelease. Needed by the search module.

Wrapper library

- Determine whether a task may be disabled by executing a potentially blocking API call.
- Label each call to the API with an appropriate triple.
- Inform the CHESS scheduler about the creating and termination of a task.

Robustness in design

- Conservatively setting isWrite to true only adds extra edges in the happens-before graph
- Conservatively setting isRelease to true might creating some wasteful work.

Code complexity

API	No. of wrappers	LOC
Win32	134	2512
.NET	64	1270
Singularity	37	876

Data-race problem

Problem

Most concurrent programs contain data-races, which cannot be captured by the wrappers.

Solutions

Enforce single-threaded execution to ensure that two threads cannot concurrently access memory locations. All data-races occur in the order in which CHESS schedules the threads.

Data-race problem

Problem

Most concurrent programs contain data-races, which cannot be captured by the wrappers.

Downsides

- Slow down the execution of the program. But this lost performance can be recovered by running multiple CHESS instances in parallel.
- CHESS may not be able to explore both of the possible outcomes of a data-race. Can be addressed by running a data-race detector.

Outline

Basic search operation

Strategy

- CHESS repeatedly executes the same test driving each iteration of the test through a different schedule

Basic search operation

Strategy

- CHESS repeatedly executes the same test driving each iteration of the test through a different schedule
- In each iteration, the scheduler works in three phases: replay, record, and search.

Basic search operation

Strategy

- CHESS repeatedly executes the same test driving each iteration of the test through a different schedule
- In each iteration, the scheduler works in three phases: replay, record, and search.
- Replay: replay a sequence of scheduling choices from a trace file.

Basic search operation

Strategy

- CHESS repeatedly executes the same test driving each iteration of the test through a different schedule
- In each iteration, the scheduler works in three phases: replay, record, and search.
- Replay: replay a sequence of scheduling choices from a trace file.
- Record: the scheduler behaves as a fair, non-preemptive scheduler. It also record the thread scheduled at each schedule point with the set of threads enabled at each point.

Basic search operation

Strategy

- CHESS repeatedly executes the same test driving each iteration of the test through a different schedule
- In each iteration, the scheduler works in three phases: replay, record, and search.
- Replay: replay a sequence of scheduling choices from a trace file.
- Record: the scheduler behaves as a fair, non-preemptive scheduler. It also record the thread scheduled at each schedule point with the set of threads enabled at each point.
- Search: determine the schedule for the next iteration.

Fail to replay a trace

Two cases

- The thread to schedule at a scheduling point is disabled.
- A scheduled thread performs a different sequence of synchronization operations than the one present in the trace.

Solutions

- Lay-initialization
- Interference from environment
- Nondeterministic call
    - random()
    - gettimeofday()

State-space explosion

Problem

Given a program with n threads that execute k atomic steps in total, it is very easy to show that the number thread interleavings grows astronomically as n^k.

Inserting preemptions prudently

Reference

Given a program with n threads that execute k steps in total, the number of interleavings with c preemptions grows with k^c.

Optimizations

Scope preemptions to code regions of interest, essentially reducing k.

- A significant portion of the synchronization operations occur in system functions.
- A large number of the synchronizations are due to accesses to volatile variables.

Monitoring

Failure catching

- Null dereferences, segmentation faults, crashes.
- User can attach other monitors such as memory-leak detectors.
- Deadlock report – whenever the set of enabled tasks becomes empty.
- Livelock report – user sets an abnormally high bound on the length of the execution.

Outline

Brief description

Input programs

Programs	LOC	max threads	max synch.	max preemp.
PLINQ	23750	8	23930	2
CDS	6243	3	143	2
STM	20176	2	75	4
TPL	24134	8	31200	2
ConcRT	16494	4	486	3
CCR	9305	3	226	2
Dryad	18093	25	4892	2
Singularity	174601	14	167924	1

Bugs found by CHESS

Programs	Total	Unk/Unk	Kn/Unk	Kn/Kn
PLINQ	1		1	
CDS	1		1	
STM	2			2
TPL	9	9		
ConcRT	4	4		
CCR	2	1	1	
Dryad	7	7		
Singularity	1		1	
Total	27	21	4	2

Outline

Conclusions

- CHESS, a systematic testing tool for finding and reproducing Heisenbugs in concurrent programs.
- Achieved by carefully exposing, controlling, and searching all interleaving nondeterminism in a concurrent system.

Thank you!