Join 8474 1107 on menti.com

(For feedback at the end of the presentation)

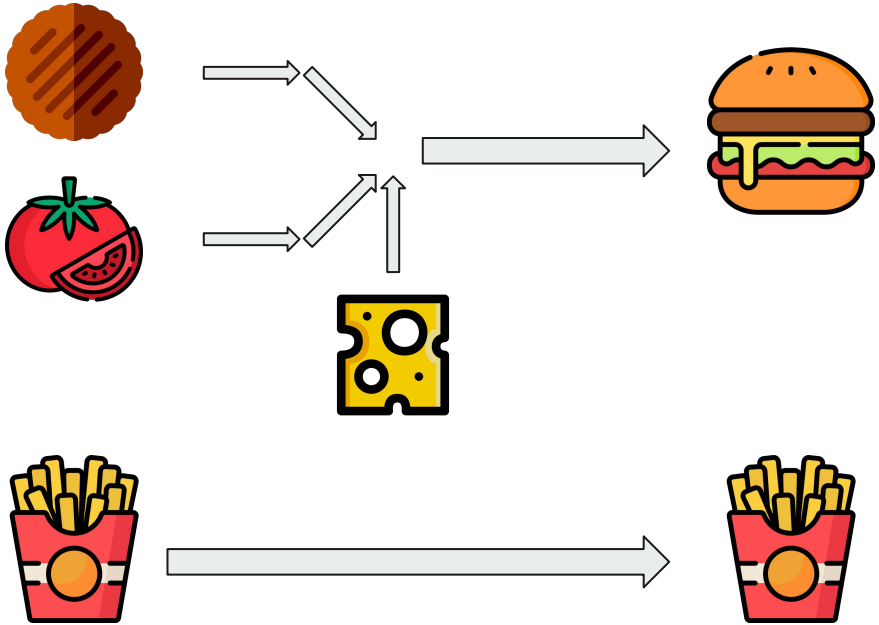
DD2482
Automated testing of Java concurrent programs with Thread Weaver

Gatien Ducornaud (gatien@kth.se)
Man Yin Edward CHUI (myechui@kth.se)

Table of contents

What is a concurrent program?

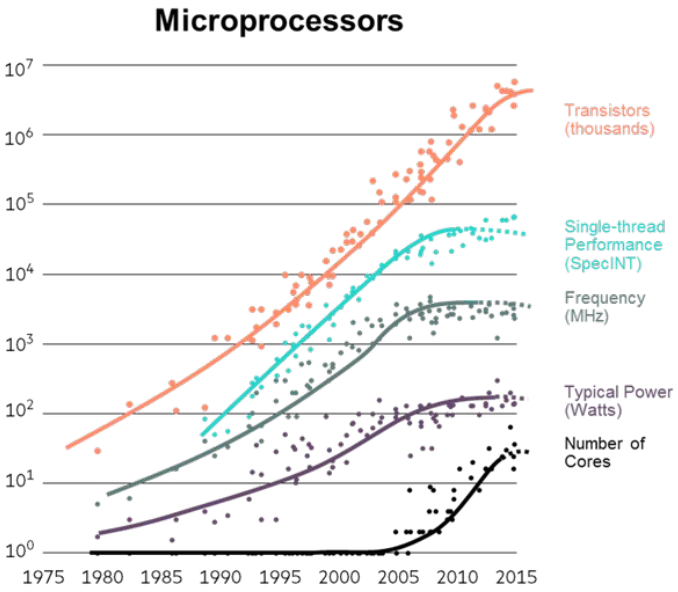
What is a concurrent program?



```java
public class BadAccount {
    private int balance = 0;

    // The keyword "synchronized" is supposed to be added here.
    public void deposit(int amount) {
        int newBalance = balance + amount;
        balance = newBalance;
    }

    public static void raceConditionDemo() {
        BadAccount account = new BadAccount();

        System.out.println("The old balance: " + account.getBalance());

        // Deposit total of 10000 into the account, by different threads.
        ExecutorService executor = Executors.newCachedThreadPool();
        for (int i = 0; i < 10000; i++) {
            executor.execute(() -> account.deposit(1));
        }
        executor.shutdown();
        // Wait for all tasks to be finished.
        while(!executor.isTerminated()) {}

        // It is expected that the new balance is 10000.
        // We can that #deposit() is not thread-safe.
        System.out.println("The new balance: " + account.getBalance());
    }
}
```

Why is it relevant?

Current landscape



Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	93,750.0	2,589

https://www.quora.com/Why-is-Moores-law-no-longer-valid

https://www.top500.org/lists/top500/list/2021/11/

Challenges of testing concurrent software

New issues

```java
public class BadAccount {
    private int balance = 0;

    // The keyword "synchronized" is supposed to be added here.
    public void deposit(int amount) {
        int newBalance = balance + amount;
        balance = newBalance;
    }

    public static void raceConditionDemo() {
        BadAccount account = new BadAccount();

        System.out.println("The old balance: " + account.getBalance());

        // Deposit total of 10000 into the account, by different threads.
        ExecutorService executor = Executors.newCachedThreadPool();
        for (int i = 0; i < 10000; i++) {
            executor.execute(() -> account.deposit(1));
        }
        executor.shutdown();
        // Wait for all tasks to be finished.
        while(!executor.isTerminated()) {}

        // It is expected that the new balance is 10000.
        // We can that #deposit() is not thread-safe.
        System.out.println("The new balance: " + account.getBalance());
    }
}
```

Race Condition/Data Race

```java
public class DeadlockExample {
    private Lock lock1 = new ReentrantLock(true);
    private Lock lock2 = new ReentrantLock(true);

    public void method1() {
        lock1.lock();
        System.out.println("Method1: Acquired lock1");

        lock2.lock();
        System.out.println("Method2: Acquired lock2");

        lock2.unlock();
        lock1.unlock();
    }

    public void method2() {
        lock2.lock();
        System.out.println("Method2: Acquired lock2");

        lock1.lock();
        System.out.println("Method2: Acquired lock1");

        lock1.unlock();
        lock2.unlock();
    }
}
```

Deadlock

Race condition

```
// The keyword "synchronized" is supposed to be added here.
public void deposit(int amount) {
    int newBalance = balance + amount;
    balance = newBalance;
}
```

Thread 1	Thread 2
newBalance = balance + amount; 0 + 1	
	newBalance = balance + amount; 0 + 1
	balance = newBalance; 1
balance = newBalance; 1	

Race condition

```java
public class BadAccount {
    private int balance = 0;

    // The keyword "synchronized" is supposed to be added here.
    public void deposit(int amount) {
        int newBalance = balance + amount;
        balance = newBalance;
    }

    public static void raceConditionDemo() {
        BadAccount account = new BadAccount();

        System.out.println("The old balance: " + account.getBalance());

        // Deposit total of 10000 into the account, by different threads.
        ExecutorService executor = Executors.newCachedThreadPool();
        for (int i = 0; i < 10000; i++) {
            executor.execute(() -> account.deposit(1));
        }
        executor.shutdown();
        // Wait for all tasks to be finished.
        while(!executor.isTerminated()) {}

        // It is expected that the new balance is 10000.
        // We can that #deposit() is not thread-safe.
        System.out.println("The new balance: " + account.getBalance());
    }
}
```

Race Condition

```
> Task :BadAccount.main()
The old balance: 0
The new balance: 9670
```

Traditional test - Stress test

```java
class BadAccountTest {
    @Test
    public void depositStressTest() {
        BadAccount account = new BadAccount();

        ExecutorService executor = Executors.newCachedThreadPool();
        for (int i = 0; i < 10000; i++) {
            executor.execute(() -> account.deposit(1));
        }
        executor.shutdown();
        // Wait for all tasks to be finished.
        while(!executor.isTerminated()) {}

        assertEquals(10000, account.getBalance());
    }
}
```

```
expected: <10000> but was: <9781>
Expected :10000
Actual   :9781
```

```
expected: <10000> but was: <9634>
Expected :10000
Actual   :9634
```

```
expected: <10000> but was: <9671>
Expected :10000
Actual   :9671
```

```
expected: <10000> but was: <9703>
Expected :10000
Actual   :9703
```

BadAccountTest.depositStressTest

Tests passed: 1 of 1 test – 48 ms

Test Results 48 ms

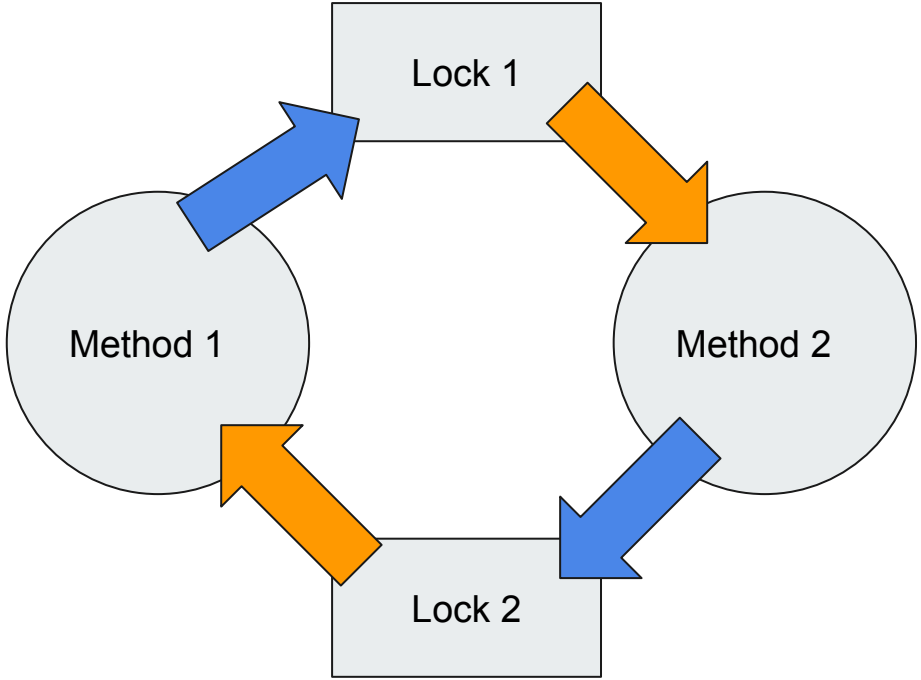
```
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
BUILD SUCCESSFUL in 2s
3 actionable tasks: 2 executed, 1 up-to-date
23:32:33: Task execution finished ':test --tests "BadAccountTest.depositStressTest"'.
```

- Reproduceable
- Repeatable
- Consistent

Deadlock

```java
public class DeadlockExample {
    private Lock lock1 = new ReentrantLock(true);
    private Lock lock2 = new ReentrantLock(true);

    public void method1() {
        lock1.lock();
        System.out.println("Method1: Acquired lock1");

        lock2.lock();
        System.out.println("Method2: Acquired lock2");

        lock2.unlock();
        lock1.unlock();
    }

    public void method2() {
        lock2.lock();
        System.out.println("Method2: Acquired lock2");

        lock1.lock();
        System.out.println("Method2: Acquired lock1");

        lock1.unlock();
        lock2.unlock();
    }
}
```

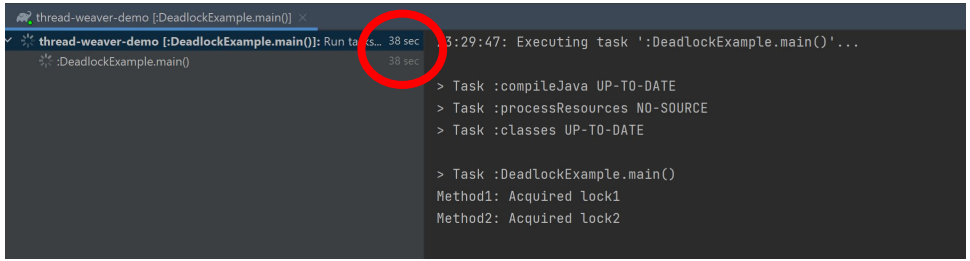
Deadlock



Deadlock

```java
public class DeadlockExample {
    private Lock lock1 = new ReentrantLock(true);
    private Lock lock2 = new ReentrantLock(true);

    public void method1() {
        lock1.lock();
        System.out.println("Method1: Acquired lock1");

        lock2.lock();
        System.out.println("Method2: Acquired lock2");

        lock2.unlock();
        lock1.unlock();
    }

    public void method2() {
        lock2.lock();
        System.out.println("Method2: Acquired lock2");

        lock1.lock();
        System.out.println("Method2: Acquired lock1");

        lock1.unlock();
        lock2.unlock();
    }
}
```

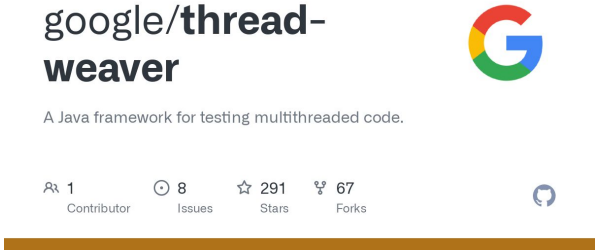
Deadlock



Introducing Thread Weaver

Thread Weaver

- Originally created by Google
  - https://github.com/google/thread-weaver
- Forked by MapDB for Java 8 compatiblity and mavenize it
  - https://github.com/jankotek/thread-weaver
- Race condition ✔️
- Thread stravation ❌
- Deadlocks❌



Buggy code

```java
// The keyword "synchronized" is supposed to be added here.
public void deposit(int amount) {
    int newBalance = balance + amount;
    balance = newBalance;
}
```

How does Thread Weaver work?

```java
public class BadAccountWithThreadWeaverTest {
    BadAccount account;

    @ThreadedBefore
    public void before() {
        account = new BadAccount();
    }

    @ThreadedMain
    public void mainThread() {
        account.deposit(10);
    }

    @ThreadedSecondary
    public void secondThread() {
        account.deposit(20);
    }

    @ThreadedAfter
    public void after() {
        assertEquals(30, account.getBalance());
    }

    @Test
    public void depositWithThreadWeaverTest() {
        AnnotatedTestRunner runner = new AnnotatedTestRunner();
        HashSet<String> methods = new HashSet<>();
        runner.setMethodOption(MethodOption.ALL_METHODS, methods);
        runner.runTests(this.getClass(), BadAccount.class);
}
```

Main Thread

Secondary Thread

2nd iteration

```java
// The keyword "synchronized" is supposed to be added here.
public void deposit(int amount) {
    int newBalance = balance + amount;
    balance = newBalance;
}
```

How does Thread Weaver work?

```
Run: BadAccountWithThreadWeaverTest
Tests failed: 1 of 1 test – 232 ms
Test Results 232 ms
  BadAccountWithThreadWeave 232 ms
    depositWithThreadWeave 232 ms

    at com.google.testing.threadtester.InterleavedRunner.interleave(InterleavedRunner.java:80)
    at com.google.testing.threadtester.AnnotatedTestWrapper.runTestCases(AnnotatedTestWrapper.java:299)
    at com.google.testing.threadtester.AnnotatedTestWrapper.runTests(AnnotatedTestWrapper.java:283)
    ... 98 more
Caused by: java.lang.reflect.InvocationTargetException <4 internal lines>
    at com.google.testing.threadtester.MethodCaller.invoke(MethodCaller.java:68)
    ... 103 more
Caused by: org.opentest4j.AssertionFailedError Create breakpoint : expected: <30> but was: <10> <5 internal lines>
    at BadAccountWithThreadWeaverTest.after(BadAccountWithThreadWeaverTest.java:28)
    ... 108 more
```

Let’s fix it

```java
public synchronized void deposit(int amount) {
        int newBalance = balance + amount;
        balance = newBalance;
}
```

Run: BadAccountWithThreadWeaverTest

Tests passed: 1 of 1 test – 371 ms

Test Results 371 ms

```
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
BUILD SUCCESSFUL in 2s
3 actionable tasks: 3 executed
23:55:35: Task execution finished ':test --tests "BadAccountWithThreadWeaverTest"'.
```

Advance features

- Enable debug message (`#setDebug(true)`)
- Finer-grained control
  - `CodePosition`
  - `Scripts`

Pros and Cons

- Allow automatic testing
- Reproducible, Repeatable, Consistent
- The only few framework that you can find

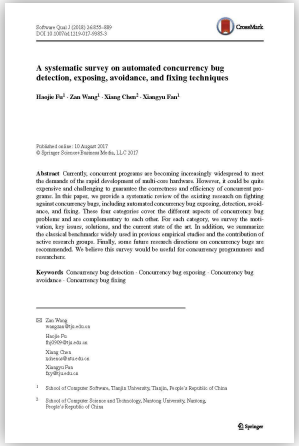
- A old framework
  - Last update was **2016** :(
- Scalability

Conclusion

Conclusion

- No silver bullet solution that can solve for everything, every system will have its own set of problems and solutions.

  So doing your own reasearch is always needed!



Software Qual J (2018) 26:955–989
DOI 10.1007/s11219-017-9385-3

A systematic survey on automated concurrency bug detection, exposing, avoidance, and fixing techniques

Haojie Fu[3] · Zan Wang[1] · Xiang Chen[2] · Xiangyu Fan[1]

Published online: 10 August 2017

Abstract Currently, concurrent programs are becoming increasingly widespread to meet the demands of the rapid development of multi-core hardware. However, it could be quite expensive and challenging to guarantee the correctness and efficiency of concurrent programs. In this paper, we provide a systematic review of the existing research on fighting against concurrency bugs, including automated concurrency bug exposing, detection, avoidance, and fixing. These four categories cover the different aspects of concurrency bug problems and are complementary to each other. For each category, we survey the motivation, key issues, solutions, and the current state of the art. In addition, we summarize the classical benchmarks widely used in previous empirical studies and the contribution of active research groups. Finally, some future research directions on concurrency bugs are recommended. We believe this survey would be useful for concurrency programmers and researchers.

Keywords Concurrency bug detection · Concurrency bug exposing · Concurrency bug avoidance · Concurrency bug fixing

✉ Zan Wang
wangzan@tju.edu.cn

Haojie Fu
fu2006@tju.edu.cn

Xiang Chen
xchen@tju.edu.cn

Xiangyu Fan
fxy@tju.edu.cn

1 School of Computer Software, Tianjin University, Tianjin, People's Republic of China

3 School of Computer Science and Technology, Nantong University, Nantong, People's Republic of China

Fu, H., Wang, Z., Chen, X., & Fan, X. (2017).
A systematic survey on automated concurrency bug detection, exposing, avoidance, and fixing techniques

All code used can be found here:

https://github.com/EDChui/thread-weaver-demo

Some feedback on this menti will be greatly appreciated!
You can also ask further questions there.

Or use code :
8474 1107
On menti.com

References

References

- https://github.com/google/thread-weaver
- https://mapdb.org/blog/thread_weaver/

For further reading on more tools:

https://link.springer.com/article/10.1007/s11219-017-9385-3