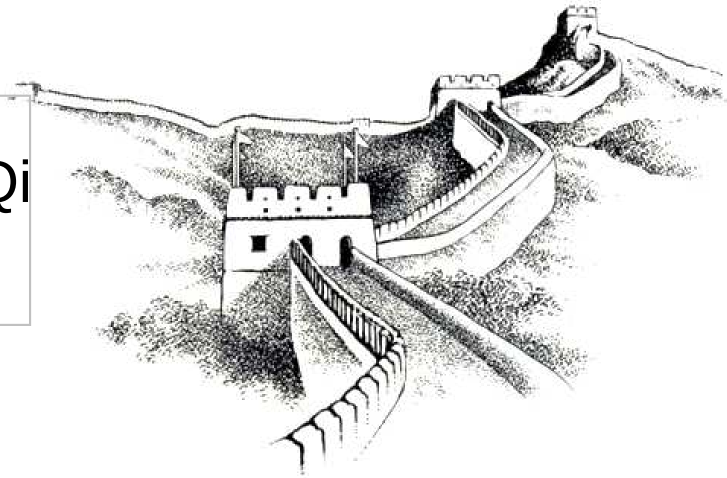
Effective static analysis to find concurrency bugs in Java

Zhi Da (Daniel) Luo, Linda Hillis, Raja Das, Yao Qi
IBM China Software Development Lab



Background

- **Multicore processors have become mainstream**
  - Need to develop concurrent software to fully exploit hardware performance
- **Difficult and error prone to write Java concurrent program**
  - Concurrency bugs often not reproducible, due to non-deterministic thread scheduling
  - Fundamental misconceptions about concurrency in Java
  - Intentionally fragile code is created to improve performance
- **Practical analysis techniques that identify concurrent bugs are valuable!**

Existing Analysis Techniques for Concurrency Bugs

- **Dynamic analysis**
  - Can reveal most concurrent bugs, such as data races, deadlock
  - Limited to finding bugs in the program paths that are actually executed
  - Incurs runtime overhead, thus prevented from running frequently
- **Model checking**
  - Systematically explores all possible thread schedules
  - Depends on the construction of a good model
  - Suffers from state-space explosion
- **Static analysis**
  - Deep analysis based on graphs
    - Gives fewer false negatives
    - Reports many false positives (infeasible paths and imprecise program information)
    - Non-scalable to large real-world applications
  - Bug patterns matching
    - Effective to find real bugs
    - Efficient analysis, scalable to large applications
    - Inaccurate, finds both false negatives and false positives

Area we focus to improve

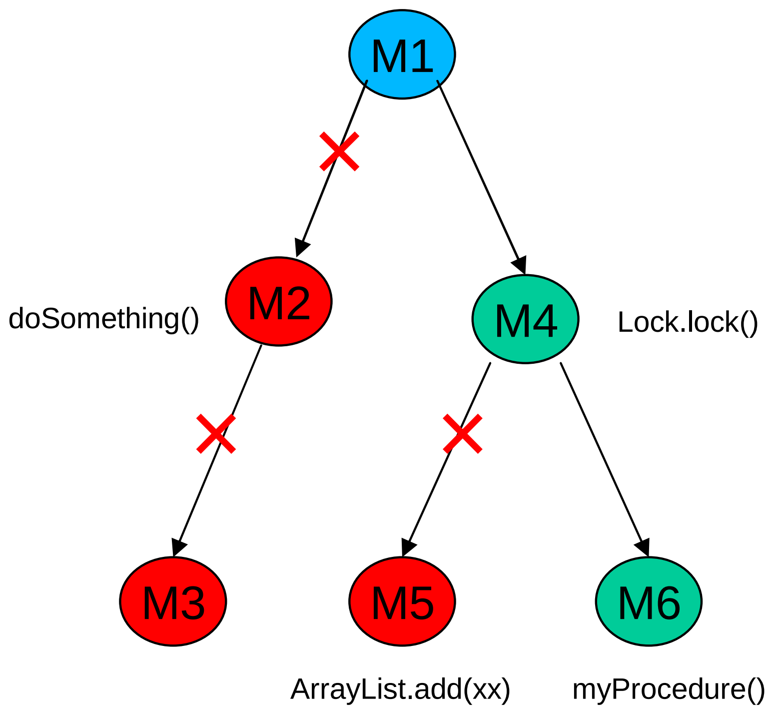
Our solution: Practical Static Concurrency Bug Patterns Detector for Real-world Applications (RSAR)

- **Define Concurrency Bug Patterns**
  - Code idioms that violate correct Java multithreaded programming practises
  - 7 commonly-seen Java multithreaded bug patterns
  - Bug pattern variants that cheat detectors

SUMMARY OF BUG PATTERNS

ID	Description
VF	Non-atomic Operations on Volatile Field Without A Lock Held
IS	Inconsistent Synchronized Monitor and Receiver of wait()/notify()/notifyAll()
LL	java.util.concurrent Lock Leak
DC	Double Checked Locking
SN	Synchronized and Null Check on The Same Field
SW	Spin Wait
SS	Synchronized Setter Method Non-synchronized Similarly-name Getter Method

- **Approaches for Different Bug Patterns**
  - Syntactically match source code with Abstract Syntax Tree
    - Novel but simple heuristics and enhancements for analysis precision and performance
    - e.g. Estimate whether a class is multithreaded or not by searching synchronization primitives
  - Inter-procedural data flow analysis based on WALA
    - For efficiency, prune call graph to include only a subset of necessary methods
    - Alias analysis using selective equality predicates without whole-program alias analysis



Accuracy & Performance

- **Accuracy**
  - Tested with 4 large real-world applications.
  - Over 65% warnings are harmful.
- **Performance**
  - Fast analysis, 16 sec to analyze 5M LOC for the slowest rule.

Experimental Environment:

Intel(R) Pentium(R) 4 CPU 2.60GHz, 1.5G memory, Windows XP Professional, RSAR 7.1.0

ID	Jetty-7.0.2 (160KLOC, 677 files)					Derby-10.5 (542KLOC, 1950 files)				
	warnings	harmful	harmless	false pos	analysis time(ms)	warnings	harmful	harmless	false pos	analysis time(ms)
VF	2	100%	0%	0%	462	0	-	-	-	1,278
IS	0	-	-	-	684	0	-	-	-	1,282
DC	1	100%	0%	0%	174	1	100%	0%	0%	521
SN	0	-	-	-	8	0	-	-	-	8
SW	0	-	-	-	190	0	-	-	-	303
SS	4	100%	0%	0%	437	20	100%	0%	0%	852
ID	Glassfish-2.1-B60 (2235KLOC, 9751 files)					Commercial Software (>5000KLOC, 19199 files)				
	warnings	harmful	harmless	false pos	analysis time(ms)	warnings	harmful	harmless	false pos	analysis time(ms)
VF	1	0%	100%	0%	3,017	1	0%	100%	0%	9,584
IS	0	-	-	-	4,400	0	-	0%	0%	16,586
DC	12	100%	0%	0%	1,401	12	100%	0%	0%	4,793
SN	0	-	-	-	29	3	67%	33%	0%	60
SW	0	-	-	-	635	3	100%	0%	0%	3,011
SS	3	100%	0%	0%	2,147	45	100%	0%	0%	7,605

Comparison with existing tool

- **RSAR (IBM)**
  - Accurate and Efficient
    - Experiment with 4 large real-world applications
    - Over 65% warnings are real bugs.
    - Slowest rule takes 16 sec to analyze 5MLOC application.
  - Inter-procedural data flow analysis

- **FindBugs (Open source)**
  - Rich set of multithreaded bug patterns
  - Fast analysis
  - Intra-procedural data flow analysis
  - Numerous false positives and false negatives
    - Linear scan through the byte code,
    - Coarse-grained code match
    - Fail to consider bug pattern variants

ID	Small test examples			Derby-10.5			Jetty-7.0.2		
	Real Bugs	RSAR	FindBugs	Real Bugs	RSAR	FindBugs	Real Bugs	RSAR	FindBugs
DC	7	7	6	1	1	2	1	1	4
SN	8	8	0	0	0	0	0	0	0
SW	11	11	2	0	0	0	0	0	0
SS	2	2	3	20	20	27	4	4	5

```
    if (streamLength != -1)
        return streamLength;

    boolean pushStack = false;
    try
    {
        // we have a stream
        synchronized (getConnectionSynchronization())
        {
            pushStack = !getEmbedConnection().isClosed();
            if (pushStack)
                setupContextStack();
```

False DCL alarms in FindBugs

```
while (!flag);
```

Typical spin wait that is not reported in FindBugs

```
synchronized (obj) {
  if (obj == null) {
    val++;
  }
}
```

Simple sync-null-check bug that is not reported in FindBugs

Bugs found in real applications

- ***Jetty***: Non-atomic self-increment operation on volatile field _set in class SelectorManager (JETTY-1187) is confirmed.

```
// Jetty 7.1.0,
// org.eclipse.jetty.io.nio,
// SelectorManager.java, line 105
private volatile int _set;
......
public void register(SocketChannel channel, Object
      att)
{
  int s=_set++;
  ......
}
......
public void addChange(Object point)
{
  synchronized (_changes)
  {
    ......
  }
}
```

- ***Derby:*** Uses an instance lock to protect static shared data in EmbedPooledConnection (DERBY-4723) is fixed.

EmbedPooledConnection has the unsafe synchronization as follow.

```
private static int idCounter = 0;

private synchronized int nextId()
  {
     return idCounter++;
  }
```

Kristian Waagan added a comment - 28/Jul/10 08:56 AM

Attached patch 1a, which removes the code using incorrect synchronization.

Kristian Waagan added a comment - 28/Jul/10 11:20 AM

Committed patch 1a to trunk with revision 980089.
Regression tests passed (12836 tests executed).

- ***Eclipse, Glassfish:*** Broken double checked locking bugs are confirmed.
  - 13 broken double checked locking bugs found in Glassfish, confirmed by community developers (Bug-11383)
  - 1 bug found in Eclipse IDE source code, confirmed and bug state was changed from “Unconfirmed” to “New” (Bug 302536)

```
    public static MarkerSupportRegistry getInstance() {
        if (singleton == null) {
            synchronized (creationLock) {
                if (singleton == null) {
                    // thread
                    singleton = new MarkerSupportRegistry();
                }
            }
        }
        return singleton;
    }
```

- ***Widely-used commercial concurrent software:***
  - Spin wait

```
                appM.uninstallApplicationLocal(
                        appName, options, this,
                        opContext.getSessionID());

                while (_waitTarget != null)
                    ; // wait for notification
```

Conclusion

- **Building an accurate and efficient Java concurrency bug patterns detector is not so difficult.**
  - Combine simple code matching analysis with novel heuristics and enhancements
  - Use inter-procedural data flow analysis with optimized techniques
- **Bug patterns detector is very effective at finding real bugs.**
- **Concurrency bugs widely exist in real-world applications!**

Controversial Statement

- **Simple analysis tools (e.g. static concurrency bug patterns detector) suffices to most software developers in practice.**

Discussion

- **Security vulnerabilities related to concurrency?**

Questions?