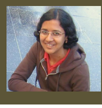
An Effective Dynamic Analysis for Detecting Generalized Deadlocks




Pallavi Joshi* Mayur Naik† Koushik Sen* David Gay‡

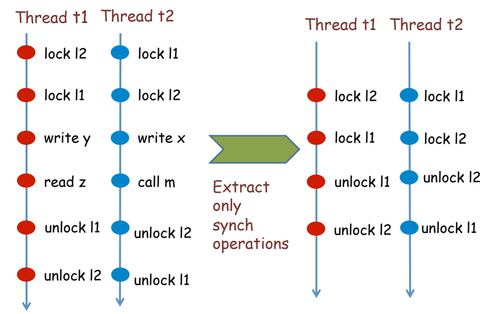
* UC Berkeley † Intel Labs Berkeley ‡ Google Inc


Motivation

- Deadlocks are a common problem in today's multithreaded software
  - 6,500/198,000 of bug reports in Sun's bug database are deadlocks
- Two categories of deadlocks
  - Resource deadlocks
  - Communication deadlocks
- Most of previous work has exclusively focused on resource deadlocks
- In this work, we address both kinds of deadlocks and also deadlocks that are hybrid between these two

Our Approach (CHECKMATE)

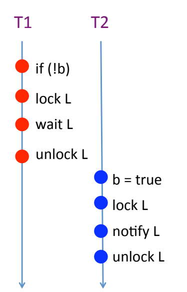
- Observe a program execution
- Retain only the synchronization operations observed during execution
  - Throw away all other operations like method invocations and memory updates
- Create a program from the retained operations (trace program)
- Model check trace program
  - Trace program is usually much more tractable to model check than the original program




```
public class TraceProgram{
    static Object l1 = new Object();
    static Object l2 = new Object();

    static Thread main = new Thread(){
        public void run(){
            t1.start(); t2.start();
        }
    };
    static Thread t1 = new Thread(){
        public void run(){
            synchronized(l2){
                synchronized(l1) {
                }
            }
    };
    static Thread t2 = new Thread(){
        public void run(){
            synchronized(l1){
                synchronized(l2) {
                }
            }
    };
    public static void main(String[] args){
        main.start();
    }
}
```

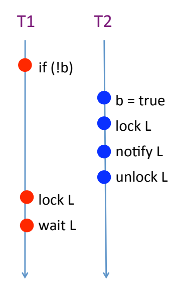
Example



Non-deadlocking interleaving

```
public class TraceProgram{
    static Object L = new Object();
    static boolean b;

    static Thread main = new Thread(){
        public void run(){
            b = false;
            t1.start(); t2.start();
        }
    };
    static Thread t1 = new Thread(){
        public void run(){
            if(!b){
                synchronized(L) {
                    L.wait();
                }
            }
    } };
    static Thread t2 = new Thread(){
        public void run(){
            b = true;
            synchronized(L) {
                L.notify();
            }
    } };
    public static void main
        (String[] args){
        main.start();
    }
}
```



Deadlocking interleaving

Implementation

- Implemented our technique in a prototype tool for Java called CHECKMATE
- Experimented with a number of Java libraries and applications
  - log4j, pool, felix, lucene, jgroups, jruby, ...
- Found both previously known and unknown deadlocks (17 in total)

Results

Program Name	# condition annotations	Original Program LOC	Trace Program LOC	Original Program Runtime	Time to generate trace program	JPF time (original program)	JPF time (trace program)	# error traces	Potential Deadlocks (#C/#R)	Confirmed Deadlocks (#C/#R)	Known Deadlocks (#C/#R)
groovy	1	45,796	59	0.118s	1s	> 1h	1.3s	5	1/0	1/0	1/0
log4j	2	48,023	225	0.116s	1s	-	8.7s	167	2/0	1/0	1/0
pool (I)	4	48,024	136	0.116s	1s	> 1h	2.3s	41	1/0	1/0	1/0
pool (II)	4	48,024	191	0.123s	1s	> 1h	2.6s	36	1/0	1/0	1/0
felix	4	73,512	113	0.173s	2.8s	-	-	-	-	1/0	1/0
lucene (I)	9	68,311	298	0.230s	3s	> 1h	1s	0	0/0	0/0	1/0
lucene (II)	9	81,071	3,534	0.296s	3.6s	> 1h	20s	0	0/0	0/0	1/0
jgroups (v1)	12	92,934	118	0.228s	4s	-	3.4s	39	2/0	1/0	1/0
jigsaw	17	122,806	3,509	-	-	-	> 1h	7894	2/7	1/5	0/2
jruby	16	136,479	966	1.1s	13.7s	-	3.9s	58	1/0	1/0	1/0
jgroups (v2)	15	160,644	2,545	9.89s	21s	-	> 1h	124	1/0	0/0	1/0
java logging	0	43,795	131	0.177s	2s	> 1h	3.7s	96	0/2	0/1	0/1
dbcp	0	90,821	400	0.74s	3.3s	-	12s	320	0/2	0/2	0/2
swing	0	264,528	1,155	0.96s	17.6s	-	105s	685	3/1	0/1	0/1

#C is no. of communication deadlocks
#R is no. of resource deadlocks

Future/Ongoing Work

- Trace program approach would also be effective for other concurrency errors that cannot be detected using an idiom
  - e.g. deadlocks because of errors or exceptions

```
// Thread 1                 // Thread 2
while (!b) {                try{
    sync (L) {                  foo();
        L.wait ();              b = true;
    }                           sync (L) { L.notify(); }
}                           } catch (Exception e) {...}
```

can throw an exception

b is initially false