MagicFuzzer: Scalable Deadlock Detection for Large-Scale Applications

Yan Cai, W.K. Chan
Introduction

• Automatic recognition of potential deadlocks
• Resource-deadlocks (non-communication)
• Dynamic method: Runs program and creates log at each critical event
• Largescale
  • Applicable for Firefox, Thunderbird...
Introduction: Lock dependency

• Dependency relation D: set of lock dependencies

• Lock dependency \(< t, m, L >\)
  • Thread \(t\)
  • Lock \(m\)
  • Lockset \(L\)
  • \(t\) holds all locks of \(L\) whilst acquiring \(m\)

• Chain of lock dependencies\(< t_1, m_1, L_1 > \ldots < t_k, m_k, L_k >\) such that every next thread holds a lock the previous tries to claim

• Deadlock: Cyclic lock dependency chain
Introduction: Lock dependency graph

• Depicts waits-for dependencies
• Node: lock
• Edge: Waits-for

\[ \text{acquire}(m_1); \text{acquire}(m_2); \]

• Deadlock:
Related work

• iGoodLock

• Direct checking on lock order graph

• Multicore SDK

• Constructs a location based lock order graph
Basic algorithm

• Generation of Execution Trace
• Magiclock
• Deadlock Confirmation & MagicScheduler
Execution Trace

• Create a log of an execution:

• For every thread creation, create a new lockset $L_i$
• Whenever acquire occurs
  • append $< t, m, L_i >$
  • $L_i = L_i \cup m$
• Whenever release occurs
  • $L_i = L_i \setminus m$
Magiclock

• Uses the log and makes a lock dependency graph from the dependencies

• How can we reduce the size of the graph?
Magiclock

• Only cycles deduct a possible deadlock

• Overfluent nodes:
  • Nodes with no edges
  • Nodes that only have outgoing or ingoing edges
  • Nodes that would be one of the above but are connected to existing ones of the above kind.
Magiclock: Categorization

• Independent-set: In- and outdegree equals 0
• Intermediate-set: In- or outdegree equals 0
• Inner-set: L only contains members of independent / intermediate set
• All others: Cyclic group; subject for possible deadlocks
Deadlock confirmation

• Given detected cycle $m_1 \ldots m_k$
• Gather relevant dependencies $< t, m_i, L >$
• Use DFS to search a cyclic dependency chain such that there is a deadlock
Deadlock confirmation: DFS

DFS(threadID, chain)
   For each ID from threadID + 1
      if(isTraversed(threadID)) continue;
      for each dependency d
         if(chain + d forms a cyclic chain) report;
         DFS(threadID, chain + d);
      endfor
   endfor
MagicScheduler

• Adapts object abstraction from DeadlockFuzzer
• Random scheduler, randomly selecting threads to advance
• At acquire of relevant threads of relevant locks: check for deadlock, pause the thread

• Thrashing: If all threads get put on hold unfavorably, thrashing may happen
• If thrashing happens, a random thread is put out of sleep
Comparison

• iGoodLock

• Direct checking on lock order graph

• Multicore SDK

• Constructs a location based lock order graph
Experiment

- Test of MagicFuzzer compared to other algorithms
- Ubuntu Linux system
- > meaning that the system crashed at the latest measure