CrystalBall: Predicting and Preventing Inconsistencies in Deployed Distributed Systems

Maysam Yabandeh, Nikola Knežević, Dejan Kostić and Viktor Kuncak

Presentor: Xiangyi Xie
Background

Inconsistency happened in a distributed system
Safety property: children and siblings are disjoint lists

Figure 2: An inconsistency in a run of RandTree
Figure 5: Finding errors by breadth-first search
CrystalBall Overview
Consistent Neighborhood Snapshots

• A snapshot consists of checkpoints of nodes’ states and it needs to be consistent.

• Running a model checking algorithm in parallel with the system.

• To collect a set of checkpoints which do not violate the happens-before relationship established by messages sent by the distributed service.
A symple mode of a distributed system
proc findConseq(currentState : G, property : (G → boolean))
  explored = emptySet(); errors = emptySet();
  localExplored = emptySet();
  frontier = emptyQueue();
  frontier.addLast(currentState);
  while (!STOP_CRITERION)
    state = frontier.popFirst();
      if (!property(state))
        errors.add(state); // predicted inconsistency found
        explored.add(hash(state));
    foreach ((n,s) ∈ state.L) // node n in local state s
      // process all network handlers
      foreach (((s,m),(s’,c)) ∈ H_M where (n,m) ∈ state.I)
        // node n handles message m according to st. machine
        addNextState(state,n,s,s’,{m},c);
      // process local actions only for fresh local states
      if (!localExplored.contains(hash(n,s)))
        foreach (((s,a),(s’,c)) ∈ H_A)
          addNextState(state,n,s,s’,{},c);
        localExplored.add(hash(n,s));
  proc addNextState(state,n,s,s’,c0,c)
    nextState.L = (state.L \ {(n,s)}) ∪ {(n,s’)};
    nextState.I = (state.I \ c0) ∪ c;
    if (!explored.contains(hash(nextState)))
      frontier.addLast(nextState);
Execution Steering

• Implementation of Execution Steering Mechanism

• Execution steering is driven by the report from the model checker, which produces a sequence of events and messages
Crystalball
Implementation Highlights
Checkpoint Manager

• enables service to collect and manage checkpoints to generate consistent neighborhood snapshots based on a notion of logical time

• Modify Mace compiler and runtime, add a snapshot on directive to the service description to inform the Mace compiler and the runtime that the service requires checkpointing.
Consequence Prediction Algorithm

Ability to replay paths previously found to lead to inconsistencies

- the node addresses in the code are assumed to be of the form 0,1,2,3,etc. a mapping from live IP addresses to model checker addresses, model checker is executing real code in the event and the message handlers

- introduced a dummy node that represents all system nodes without checkpoints in the current snapshot. All messages sent to such nodes are redirected to the dummy node
Mitigating the impact of limited-size snapshots.

- executing state machine handlers involves executing the handler in a copy of the state machine’s virtual memory (via fork()), and holding the transmission of messages until the successful completion of the consistency check.

- Example: Upon encountering an inconsistency in the copy, the runtime can simply throw it away and not execute the handler in the primary state Machine.
Replaying Past Erroneous Paths

• To ascertain that an inconsistency can still occur from the current snapshot, we replay past erroneous paths. Strictly replaying a sequence of events and messages that form a path on a new a neighborhood snapshot might be incorrect.

• Example: some messages could have only been generated by the old state checkpoint and are inconsistent with new state. Our replay technique therefore replays only timer and application events, and relies on distributed service code to generate any messages.
Event Filtering for Execution steering

- the CrystalBall controller installs an event filter into the runtime.
- Example:
  - network messages: a message type, message source and the destination
  - a local timer, event or application call, the filter just contains the identity of the handler that handles the event
Checking Safety of Event Filters

- upon encountering an inconsistency, we allow consequence prediction to pursue actions that an event filter could perform.
Evaluation

• Is CrystalBall effective in finding bugs in live runs?
• Can any of the bugs found by CrystalBall also be identified by the MaceMC model checker alone?
• Is execution steering capable of avoiding inconsistencies in deployed distributed systems?
• Are the overheads introduced by CrystalBall within acceptable levels?
Deep Online Debugging

<table>
<thead>
<tr>
<th>System</th>
<th>Bugs found</th>
<th>LOC Mace/C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandTree</td>
<td>7</td>
<td>309 / 2000</td>
</tr>
<tr>
<td>Chord</td>
<td>3</td>
<td>254 / 2200</td>
</tr>
<tr>
<td>Bullet’</td>
<td>3</td>
<td>2870 / 19628</td>
</tr>
</tbody>
</table>

Using CrystalBall, summary of inconsistencies found in 3 systems: RandTree, Chord and Bullet’. LOC stands for lines of code and reflects both the MACE code size and the generated C++ code size. The low LOC counts for Mace service implementations are a result of Mace’s ability to express these services succinctly.
Example RandTree Bug Found

An inconsistency in a run of a RandTree, Root(9) appears as a child
An inconsistency in a run of Chord. Node C has its predecessor pointing to itself while its successor list includes other nodes.
Example Bullet’ Bug Found

• The Bullet’source sends the blocks of the file to a subset of nodes in the system; other nodes discover and retrieve these blocks by explicitly requesting them.
• Sender’s file map and receivers view of it should be identical.
• *Scenario exhibiting inconsistency*
• Consequence of the inconsistency
Comparison with MaceMC

Illustrates the performance of MaceMC when is used for exhaustive search
conclusion

• Due to the MaceMC’s search strategy, it had difficulty advancing in the 100-node case
• Subsequently did not identify any violations
• This result validates our approach and confirms the importance of online model checking from a current, consistent neighborhood snapshot for bug finding.
Execution Steering Experience

evaluate the capability of CrystalBall as a runtime mechanism for steering execution away from previously unknown bugs
RandTree Execution Steering

A
Propose \((A_0, 0)\)
Promise \((A_0, 0)\)
Accept \((A_0, 0)\)
Learn \((A_0, 0)\)

B
Propose \((A_0, 0)\)
Promise \((A_0, 0)\)
Accept \((A_0, 0)\)
Learn \((A_0, 0)\)

C
Propose \((B_0, 1)\)
Promise \((B_0, 1)\)
Accept \((B_0, 1)\)
Learn \((B_0, 1)\)

C is disconnected
C is reachable

0 is chosen
A is disconnected

1 is chosen (instead of 0)
Figure 13: In 200 runs that expose Paxos safety violations due to two injected errors, CrystalBall successfully avoided the inconsistencies in all but 2 and 5 cases, respectively.
Performance Impact of CrystalBall

Memory
CPU
Bandwidth consumption

Conclusions:
• RandTree: does not adversely affect the performance of it
• Bullet: Having CrystalBall monitor a bandwidth-intensive such as Bullet’ does not significantly impact the application’s performance
Crystalball VS. MaceMC

- MaceMC represents in model checking distributed system implementations. MaceMC runs state machines for multiple nodes within the same process, and can determine safety and liveness violations spanning multiple nodes. MaceMC’s exhaustive state exploration algorithm limits in practice the search depth and the number of nodes that can be checked.
- In contrast, CrystalBall’s consequence prediction allows it to achieve significantly shorter running times for similar depths, thus enabling it to be deployed at runtime.
Conclusion

• Crystalball: a new approach for improving the reliability of distributed systems, where nodes predict and avoid inconsistencies before they occur, even if they have not manifested in any previous run, and can steer execution away from inconsistencies at runtime.
Any questions?