If At First You Don't Succeed, Try, Try, Again...?

Insights and LLM-informed Tooling for Detecting Retry Bugs in Software Systems

Bogdan Stoica, Utsav Sethi, Yiming Su, Cyrus Zhou, Shan Lu, Jonathan Mace, Madanlal Musuvathi, Suman Nath

THE UNIVERSITY OF CHICAGO

Microsoft Research

Motivation

- Retry is a common defense against faults and configuration problems at run-time
- Broken fault handling (incl. retry) contribute to substantial portion of cloud incidents
- Testing retry correctness is challenging: reqs simulating transient errors and specialized assertions
- Existing bug-detection techniques are not tailored to specific characteristics of retry bugs

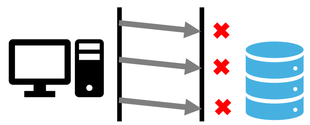
What are causes and symptoms of real-world retry issues?

How can we fine-tune/augment program analysis techniques for targeted retry-bug detection?

Study results

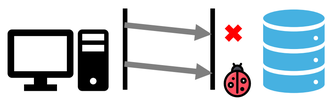
Classify **70 retry-related issues** in **9 open-source applications**

Bad policies (2/3 of bugs)



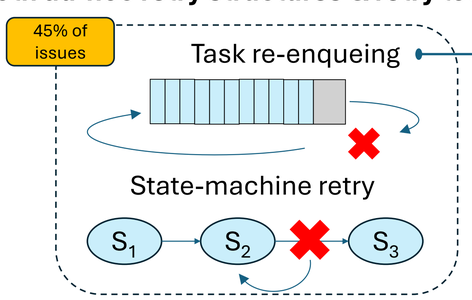
- Missing retry caps
- Missing retry delays
- Retrying non-recoverable errors
- Not retrying recoverable errors
- & others

Broken mechanisms (1/3 of bugs)



- Wrongly maintained task metadata
- Incorrectly revert system state

Both ad-hoc retry structures & retry loops are common in applications

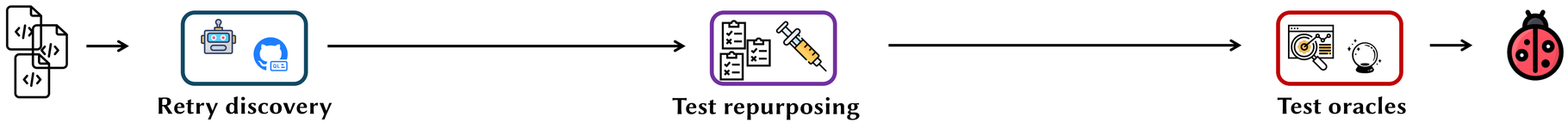



```
class CommitResponseHandler {
  void handle(Error e, Future future) {
    if (e == COORDINATOR_LOAD_IN_PROGRESS ||
    +   e == UNKNOWN_TOPIC_OR_PARTITION
    ) {
        future.raise(RetryException());
        return;
    } else {
        future.raise(DoNotRetryException());
        return;
    }
  }
}

class ConsumerCoordinator {
    void commit() {
        ...
        sendCommit(msg, new CommitResponseHandler())
    }
}
```

- Can we ID ad-hoc structures in source code for targeted analysis / detection?
- How to distinguish retry loops (other 55% of issues) from non-retry loops?

WASABI: a toolkit to find retry bugs



Identify retry-code locations using combined structure + context analysis via CodeQL and GPT

CodeQL (loop locations)

```
predicate hasReentryFromCatch(LoopStmt loop) {
    exists(CatchClause cc | isCaughtTopLevel(loop, cc) and
           exists(Expr loopReentry, ControlFlowNode last |
               if exists(loop.(ForStmt).getAnUpdate())
               then loopReentry = loop.(ForStmt).getUpdate(0)
               else loopReentry = loop.getCondition()
           |
               last.getEnclosingStmt().getEnclosingStmt*() = cc.getBlock()
               and
               hasSuccessor(last, loopReentry)
           ))
}
```

Combine **structural patterns** (e.g. re-entry on exception) with **keyword filtering** (log messages, variable names) to distinguish retry from non-retry **loops**

GPT-4 (loop + non-loop locations)

```
Does the following code perform retry anywhere
? Answer (Yes) or (No).
  - Say NO if the file only _defines_ or _creates_
      retry policies, or only passes retry
      parameters to other builders/constructors.
  - Say NO if the file does not check for
      exception or errors before retry.
  **Remember that retry mechanisms can be
      implemented through "for" or "while" loops
      or data structures like state machines and
      queues.**
```

- Hints to guide model towards non-standard retry modes
- Pinpoint location to retry-containing method
- Exclude false positives (e.g. polling, retry-configs)

List of (method, retry-causing-exception) tuples for each location
(300+ retry locs across 8 apps)

Inject faults during test runs at locations identified in previous step

HDFSInputStream.java

```
bool createBlockReader():
 1. while (true) {
 2.   try {
 3.     block = refreshBlock(block);
 4.     dnInfo = getDNInfFor(block);
 5.     if (dnInfo == null)
 6.       break;
 7.   }
 8.   catch (IOException e) {
 9.     LOG("Failed to connect to "+
          dnInfo.addr + "Retried " +
            ++retryCount + " times");
10.    addToDeadNodes(dnInfo.info);
11.  }
12. }
```

Retried method/ex. identified by discovery phase

- Instrument (retried method, exception) using AspectJ
- Inject retry-triggering exception *N* times during test runs

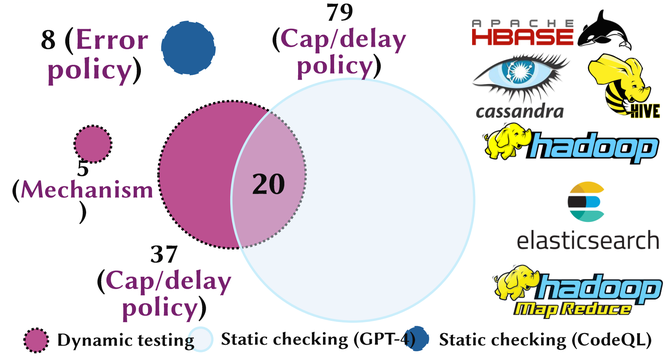
- Test-planning phase distributes conflicting retry executions across multiple unit tests

Analyze/monitor test output using different oracles to pinpoint retry bugs

Different exception oracle	• Flag cases where failure-causing exception is different from retry-triggering exception • Captures diverse retry failure modes (broken mechanisms)
Missing cap oracle	• Flag cases that continue beyond *N* attempts (bad policies)
Missing delay oracle	• Flag cases that do not perform delay-related calls between executions (bad policies)

Additional static checking workflows to directly infer additional policy bugs from source code..see paper for details!

Bug detection results



- **109 retry bugs** found across **8 open-source systems**
- Typically <12h per system
- $8 median **GPT cost** per system
- 1 **false** per every 2 **bugs** found

Conclusions

- Retry is necessary to handle transient failures
- We introduce WASABI, a novel suite of techniques to detect retry bugs using repurposed unit testing & static checking
- **Check out our paper at SOSP and our artifact!** https://github.com/bastoica/wasabi