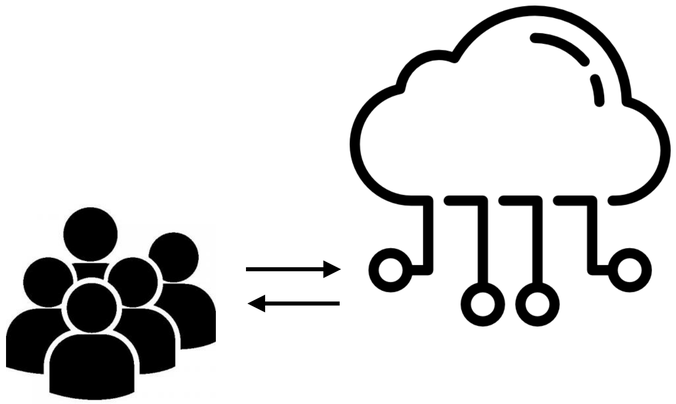
Deriving Semantic Checkers from Tests to Detect Silent Failures in Production Distributed Systems

Chang Lou, Dimas Shidqi Parikesit, Yujin Huang, Zhewen Yang,
Senapati Diwangkara, Yuzhuo Jing, Achmad Imam Kistijantoro,
Ding Yuan, Suman Nath, Peng Huang

Rich semantics in distributed systems

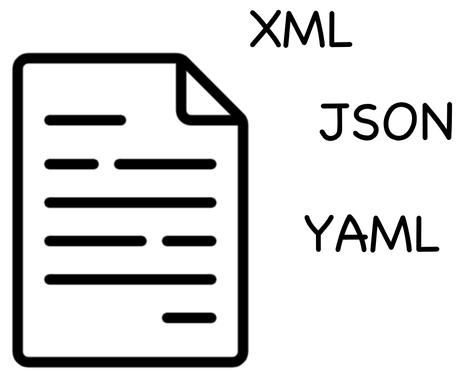


Client APIs

watch, prune, rec..ct..

Component guarantees

messa..dering, redund.., ACID..

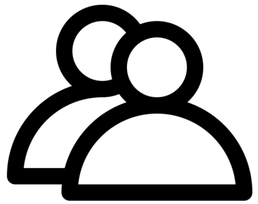


Configurations

tickTime..pCount, maxCl..nxns..

Silent failure: semantics gets violated without error signals

Example: ZooKeeper ephemerals



Services **ZooKeeper** **Clients**

1 

service1: addr1 → service1: addr1

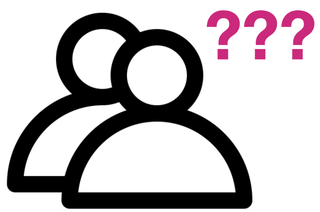
2 

<empty> → UnavailableException

non-silent failures

Guarantee: Zookeeper remove ephemerals when sessions disconnect

Example: ZooKeeper ephemerals



Services | **ZooKeeper** | **Clients**

1. service1: addr1 → service1: addr1
2. service1: addr1 → service1: addr1

silent failures

Silent failures in real-world

Networking issues take down Google Cloud in parts of the U.S. and Europe, YouTube and Snapchat also affected

, the bug caused these jobs to incorrectly throttle outbound network traffic

TECH

Google issues apology, incident report for hourslong cloud outage

the reported usage was inaccurately being reported as zero,

Cloud ▶

6 Things To Know About The Latest Salesforce Outage

How to Find Silent Failures in Your Cloud Services Faster

tions about the worst service disruption in
know and what we would still like to find

stela udovicic
September 30, 2019

Silent Data Corruption at Scale

by Caroline Trippel on Aug 15, 2022 | Tags: Datacenters, Errors, Reliability, Testing

Cores that don't count

Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul — Google, Sunnyvale, CA, US

Rama Govindaraju, Parthasarathy Ranganathan — Google, Sunnyvale, CA, US

David E. Culler, Amin Vahdat — Google, Sunnyvale, CA, US

Understanding Silent Data Corruptions in a Large Production CPU Population

Shaobu Wang — Tsinghua University

Guangyan Zhang* — Tsinghua University

Junyu Wei — Tsinghua University

Yang Wang — The Ohio State University

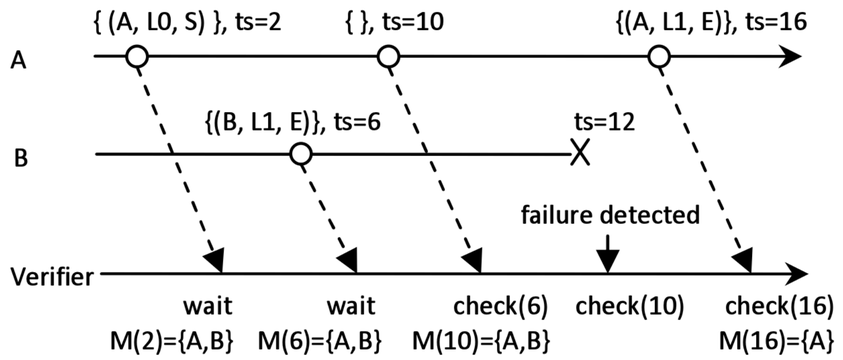
Jiesheng Wu — Alibaba Cloud

Qingchao Luo — Alibaba Cloud

There is a pressing need for runtime verification with semantic checkers!

Manually writing semantic checkers is not easy

- Prior works often focus on well-defined properties for protocols.
  - we target abundant loosely-defined semantics
- Require deep understandings of system codes and behaviors
  - i.e., checking target, expectation, timing



D3S [NSDI'08]

```
1  Define DefinedMemory(ot) as:
2  Select m.rVal as pointer, m.exitTime as start,
3         m.callStack as allocSite, f.enterTime as end
4         m.arg["size"] as size
5  From Funcs(ot) as m Left NextJoin Functs(ot) as f
6    On m.exitTime, f.enterTime,
7    m.name="malloc" And m.exitTime<=f.enterTime
8   And f.name="free" And m.rVal=f.arg["ptr"]
9
10 Select start, allocSite, sum(size)
11   Over(Partition By allocSite Order By start)
12 From DefinedMemory(ot)
13 Where end=NULL
```

OmniTable [OSDI'22]

Automatically deriving semantic checkers from tests

```
public void testSessionTimeout() {
  ZooKeeper zk = createClient(TIMEOUT);
  zk.create("/stest", ..., EPHEMERAL);
  zk.close();
  zk.disconnect();
  zk = createClient(TIMEOUT);
  Assert.assertTrue(
      zk.exists("/stest",false) != null);
  Thread.sleep(TIMEOUT*2);
  Assert.assertTrue(
      zk.exists("/stest",false) == null);
  zk.close();
  }
```

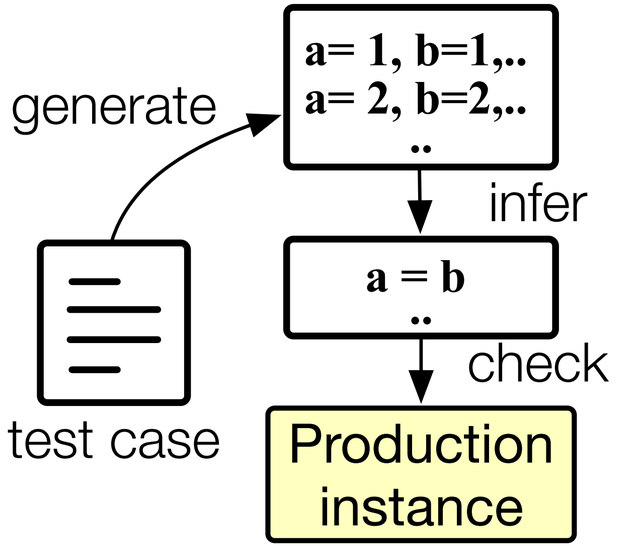
?

Test case

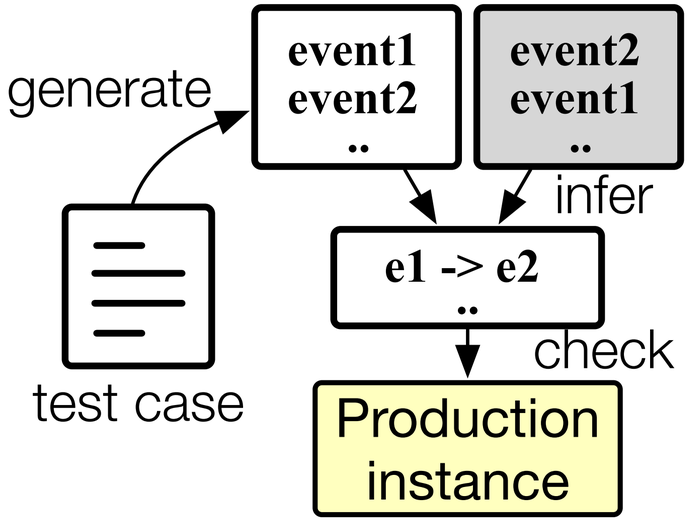
Runtime checker

Prior works: inferring invariants from tests

- Limitation 1: inferred state/event relations are too simple and not expressive
- Limitation 2: inaccuracies in statistical inference



Daikon [ICSE'99]
Dinv [ICSE'18]

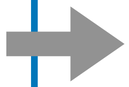


OathKeeper [OSDI'22]

Our approach: systematically transform tests to semantic checkers

```
public void testSessionTimeout() {
  ZooKeeper zk = createClient(TIMEOUT);
  zk.create("/stest", ..., EPHEMERAL);
  zk.close();
  zk.disconnect();
  zk = createClient(TIMEOUT);
  Assert.assertTrue(
      zk.exists("/stest",false) != null);
  Thread.sleep(TIMEOUT*2);
  Assert.assertTrue(
      zk.exists("/stest",false) == null);
  zk.close();
  }
```

generalized checker precondition



parameterized checker function

Test case

Runtime checker

Feasibility study

- We study 210 test cases in six popular distributed systems.
- For each test, we ask
  - (1) do tests contain useful semantic checks? how is the checking done?
  - (2) can the checking logic be generalized from tests to production settings?
  - ... *more in the full paper*

Software	Language	Category	Version	Tests (total)	Tests (sampled)
ZooKeeper	Java	Coordination	3.4.11	434	35
Cassandra	Java	Database	3.11.5	4118	35
HDFS	Java	File Sys.	3.2.2	3265	35
MongoDB	C++	Database	7.1.0	1276	35
CephFS	C++	File Sys.	1.11.0	506	35
Mesos	C++	Cluster Mgr.	18.2.0	233	35

Study findings (selected)

- Finding 1: Most (87%) tests contain explicit assertions for semantics
  - Modern software prepared many utility functions in tests

```
class DFSTestUtil {
  boolean checkFiles(FileSystem fs, String topdir)
  long verifyExpectedCacheUsage(long
    expectedCacheUsed, long expectedBlocks,
    FsDatasetSpi<?> fsd)
  void checkComponentsEquals(byte[][] expected,
    byte[][] actual)
  String runFsck(Configuration conf, int
    expectedErrCode, boolean checkErrorCode,
    String... path)
    ...
```

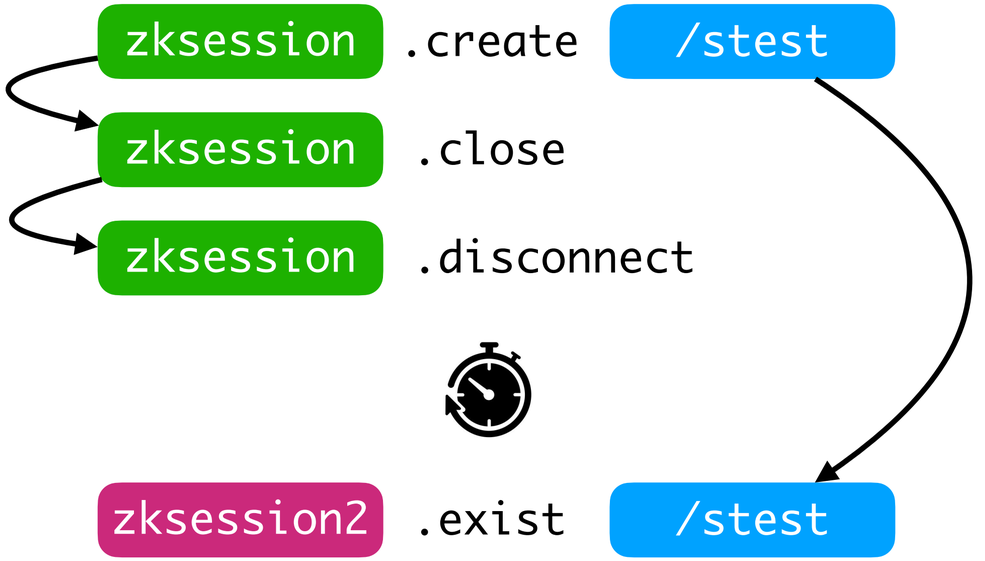
A test utility class from HDFS

Study findings (selected)

- Finding 2: Most (78%) tests have simple relations between assertion and workloads
  - in 66% of tests, checking logic can be generalized to production

```
class DFSTestUtil {
  boolean checkFiles(FileSystem fs, String topdir)
  long verifyExpectedCacheUsage(long
    expectedCacheUsed, long expectedBlocks,
    FsDatasetSpi<?> fsd)
  void checkComponentsEquals(byte[][] expected,
    byte[][] actual)
  String runFsck(Configuration conf, int
    expectedErrCode, boolean checkErrorCode,
    String... path)
    ...
```

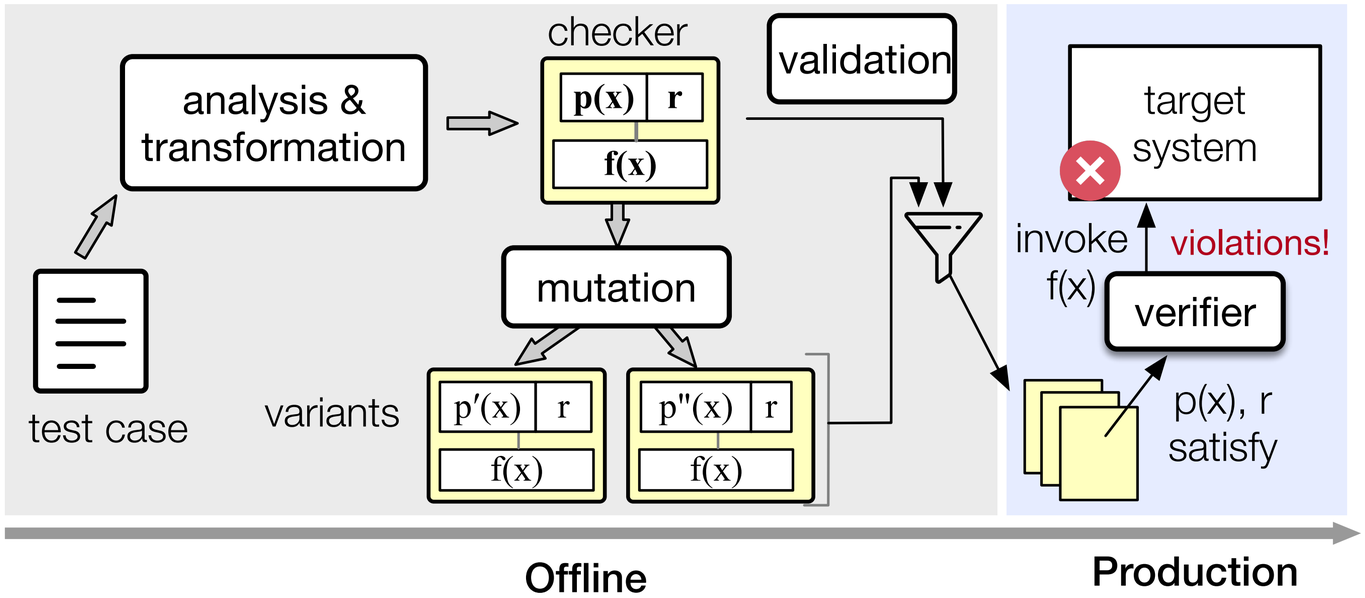
A test utility class from HDFS



State relations in ZooKeeper test traces

System workflow

- T2C: systematically derive semantic checkers from tests
  - encapsulate checker functions from mixed test logic
  - symbolize workloads to extract generalized preconditions



#1 Encapsulate checker function with static slicing

```
1  public void testSnapshotfileLength() {
2    Path file1 = new Path(sub, file1Name);
3    DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
4          0, BLOCKSIZE, REPLICATION, SEED);
5    DFSTestUtil.appendFile(hdfs, file1, origLen);
6    hdfs.allowSnapshot(sub);
7    hdfs.createSnapshot(sub, snapshot1);
8
9    Path file1snap1 = SnapshotTestHelper.
10       getSnapshotPath(sub, snapshot1, file1Name);
11   FSDataOutputStream out = hdfs.append(file1);
12   AppendTestUtil.write(out, 0, toAppend);
13   byte[] dataFromSnapshot = DFSTestUtil.
14       readFileBuffer(hdfs, file1snap1);
15 → assertThat("Wrong data size in snapshot.",
16       dataFromSnapshot.length, is(origLen));
17 }
```

Test for HDFS snapshots

Semantics: HDFS snapshot is immutable—file length remains unchanged after append.

#1 Encapsulate checker function with static slicing

```
1  public void testSnapshotfileLength() {
2    Path file1 = new Path(sub, file1Name);
3    DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
4          0, BLOCKSIZE, REPLICATION, SEED);
5    DFSTestUtil.appendFile(hdfs, file1, origLen);
6    hdfs.allowSnapshot(sub);
7    hdfs.createSnapshot(sub, snapshot1);
8
9    Path file1snap1 = SnapshotTestHelper.
10       getSnapshotPath(sub, snapshot1, file1Name);
11   FSDataOutputStream out = hdfs.append(file1);
12   AppendTestUtil.write(out, 0, toAppend);
13   byte[] dataFromSnapshot = DFSTestUtil.
14       readFileBuffer(hdfs, file1snap1);
15 → assertThat("Wrong data size in snapshot.",
16       dataFromSnapshot.length, is(origLen));
17 }
```

Test for HDFS snapshots

Semantics: HDFS snapshot is immutable—file length remains unchanged after append.

#1 Encapsulate checker function with static slicing

```
1  public void testSnapshotfileLength() {
2    Path file1 = new Path(sub, file1Name);
3    DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
4          0, BLOCKSIZE, REPLICATION, SEED);
5    DFSTestUtil.appendFile(hdfs, file1, origLen);
6    hdfs.allowSnapshot(sub);
7    hdfs.createSnapshot(sub, snapshot1);
8
9    Path file1snap1 = SnapshotTestHelper.
10       getSnapshotPath(sub, snapshot1, file1Name);
11   FSDataOutputStream out = hdfs.append(file1);
12   AppendTestUtil.write(out, 0, toAppend);
13   byte[] dataFromSnapshot = DFSTestUtil.
14       readFileBuffer(hdfs, file1snap1);
15 → assertThat("Wrong data size in snapshot.",
16       dataFromSnapshot.length, is(origLen));
17 }
```

Test for HDFS snapshots

Semantics: HDFS snapshot is immutable—file length remains unchanged after append.

#1 Encapsulate checker function with static slicing

```
1  public void testSnapshotfileLength() {
2    Path file1 = new Path(sub, file1Name);
3    DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
4          0, BLOCKSIZE, REPLICATION, SEED);
5    DFSTestUtil.appendFile(hdfs, file1, origLen);
6    hdfs.allowSnapshot(sub);
7    hdfs.createSnapshot(sub, snapshot1);
8
9    Path file1snap1 = SnapshotTestHelper.
10       getSnapshotPath(sub, snapshot1, file1Name);
11   FSDataOutputStream out = hdfs.append(file1);
12   AppendTestUtil.write(out, 0, toAppend);
13   byte[] dataFromSnapshot = DFSTestUtil.
14       readFileBuffer(hdfs, file1snap1);
15 → assertThat("Wrong data size in snapshot.",
16       dataFromSnapshot.length, is(origLen));
17 }
```

Test for HDFS snapshots

Semantics: HDFS snapshot is immutable—file length remains unchanged after append.

#1 Encapsulate checker function with static slicing

```
public void testSnapshotfileLength() {
  Path file1 = new Path(sub, file1Name);
  DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
        0, BLOCKSIZE, REPLICATION, SEED);
  DFSTestUtil.appendFile(hdfs, file1, origLen);
  hdfs.allowSnapshot(sub);
  hdfs.createSnapshot(sub, snapshot1);

  Path file1snap1 = SnapshotTestHelper.
      getSnapshotPath(sub, snapshot1, file1Name);
  FSDataOutputStream out = hdfs.append(file1);
  AppendTestUtil.write(out, 0, toAppend);
  byte[] dataFromSnapshot = DFSTestUtil.
      readFileBuffer(hdfs, file1snap1);
  assertThat("Wrong data size in snapshot.",
      dataFromSnapshot.length, is(origLen));
}
```

Test for HDFS snapshots

Special handling

- Side-effectful operation
- Identifying intermediate variables
- Multiple assertions

```
public static void generatedAssertionFunc0 (
    FileSystem hdfs, Path sub, String snapshot1,
    String file1Name, Integer origLen) {
+ Assertion.appendCustomAssert(...);

  Path file1snap1 = SnapshotTestHelper.
      getSnapshotPath(sub, snapshot1, file1Name);
  byte[] dataFromSnapshot = DFSTestUtil.
      readFileBuffer(hdfs, file1snap1);
  assertThat("Wrong data size in snapshot.",
          dataFromSnapshot.length, is(origLen));
}
```

Checker function

#1 Encapsulate checker function with static slicing

```
public void testSnapshotfileLength() {
  Path file1 = new Path(sub, file1Name);
  DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
        0, BLOCKSIZE, REPLICATION, SEED);
  DFSTestUtil.appendFile(hdfs, file1, origLen);
  hdfs.allowSnapshot(sub);
  hdfs.createSnapshot(sub, snapshot1);

  Path file1snap1 = SnapshotTestHelper.
      getSnapshotPath(sub, snapshot1, file1Name);
  FSDataOutputStream out = hdfs.append(file1);
  AppendTestUtil.write(out, 0, toAppend);
  byte[] dataFromSnapshot = DFSTestUtil.
      readFileBuffer(hdfs, file1snap1);
  assertThat("Wrong data size in snapshot.",
      dataFromSnapshot.length, is(origLen));
}
```

Test for HDFS snapshots

How to calculate parameters in production instances?

```
public static void generatedAssertionFunc0 (
    FileSystem hdfs, Path sub, String snapshot1,
    String file1Name, Integer origLen) {
+ Assertion.appendCustomAssert(...);

  Path file1snap1 = SnapshotTestHelper.
      getSnapshotPath(sub, snapshot1, file1Name);
  byte[] dataFromSnapshot = DFSTestUtil.
      readFileBuffer(hdfs, file1snap1);
  assertThat("Wrong data size in snapshot.",
            dataFromSnapshot.length, is(origLen));
}
```

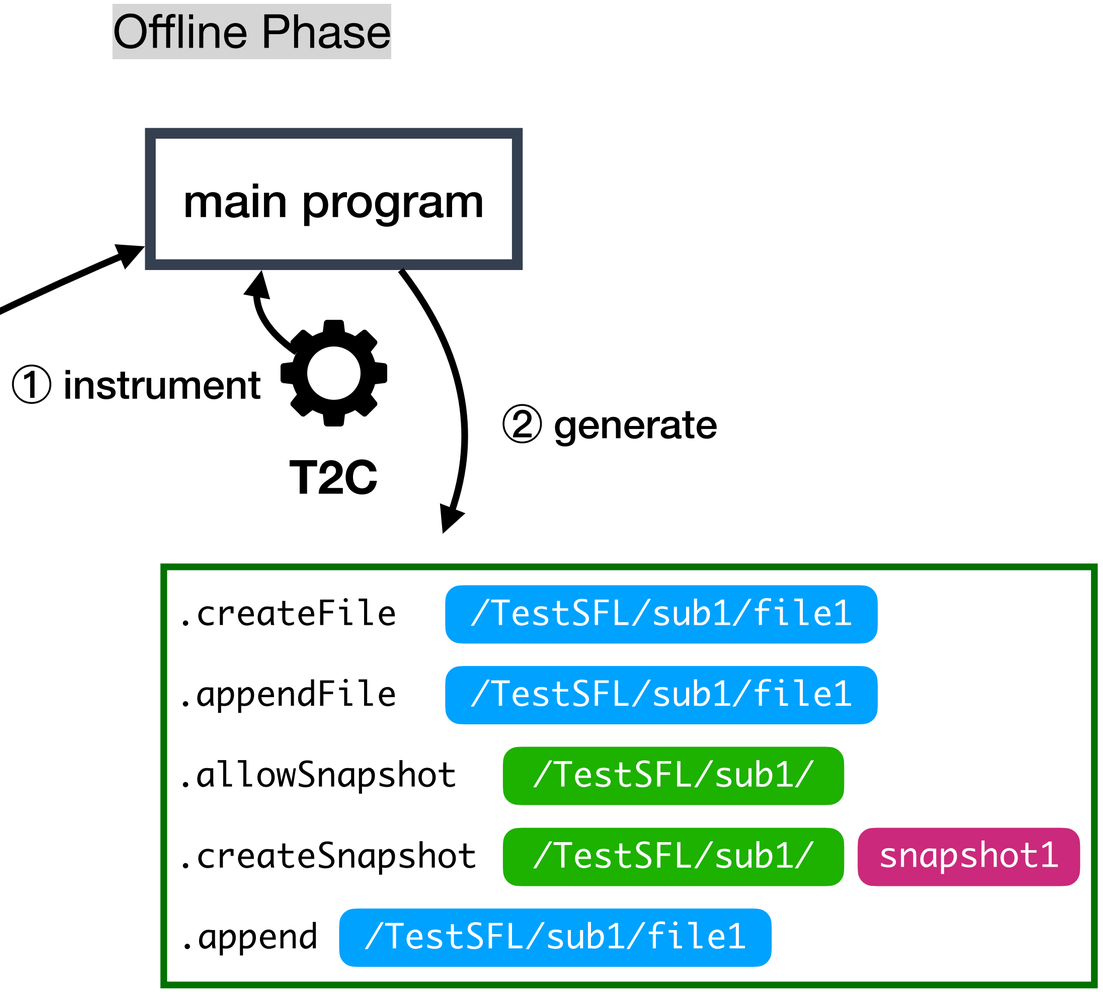
Checker function

#2 Identify checker precondition with symbolization

```
public void testSnapshotfileLength() {
  Path file1 = new Path(sub, file1Name);
  DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
        0, BLOCKSIZE, REPLICATION, SEED);
  DFSTestUtil.appendFile(hdfs, file1, origLen);
  hdfs.allowSnapshot(sub);
  hdfs.createSnapshot(sub, snapshot1);

  Path file1snap1 = SnapshotTestHelper.
      getSnapshotPath(sub, snapshot1, file1Name);
  FSDataOutputStream out = hdfs.append(file1);
  AppendTestUtil.write(out, 0, toAppend);
  byte[] dataFromSnapshot = DFSTestUtil.
      readFileBuffer(hdfs, file1snap1);
  assertThat("Wrong data size in snapshot.",
      dataFromSnapshot.length, is(origLen));
}
```

Test for HDFS snapshots



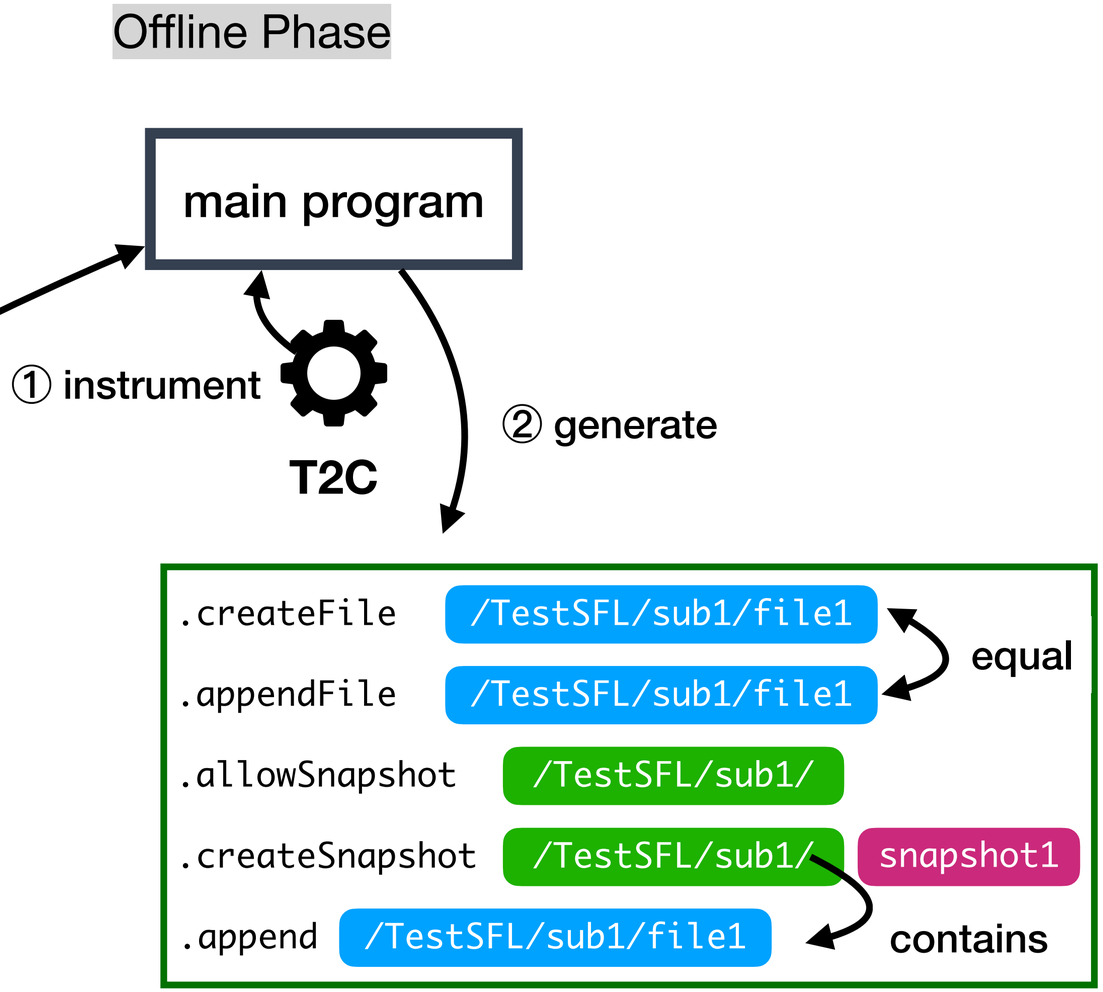
Precondition

#2 Identify checker precondition with symbolization

```
public void testSnapshotfileLength() {
  Path file1 = new Path(sub, file1Name);
  DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
        0, BLOCKSIZE, REPLICATION, SEED);
  DFSTestUtil.appendFile(hdfs, file1, origLen);
  hdfs.allowSnapshot(sub);
  hdfs.createSnapshot(sub, snapshot1);

  Path file1snap1 = SnapshotTestHelper.
      getSnapshotPath(sub, snapshot1, file1Name);
  FSDataOutputStream out = hdfs.append(file1);
  AppendTestUtil.write(out, 0, toAppend);
  byte[] dataFromSnapshot = DFSTestUtil.
      readFileBuffer(hdfs, file1snap1);
  assertThat("Wrong data size in snapshot.",
      dataFromSnapshot.length, is(origLen));
}
```

Test for HDFS snapshots



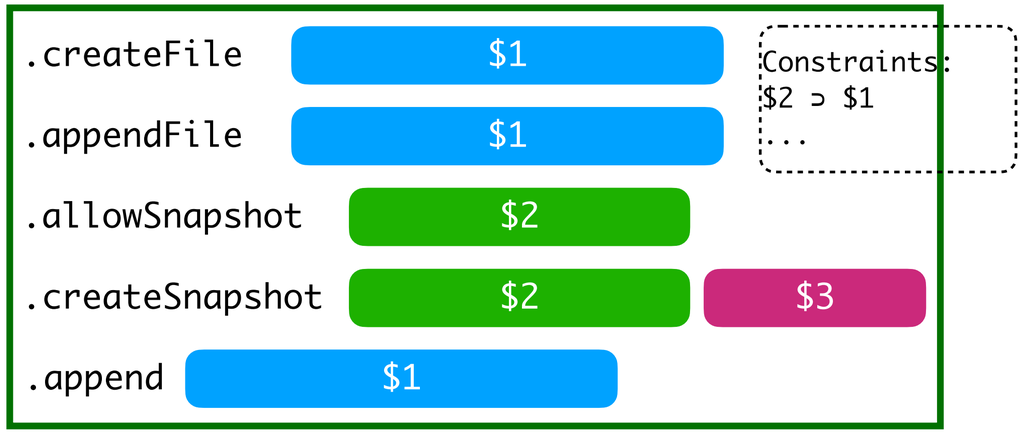
Precondition

#2 Identify checker precondition with symbolization

```
public void testSnapshotfileLength() {
  Path file1 = new Path(sub, file1Name);
  DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
        0, BLOCKSIZE, REPLICATION, SEED);
  DFSTestUtil.appendFile(hdfs, file1, origLen);
  hdfs.allowSnapshot(sub);
  hdfs.createSnapshot(sub, snapshot1);

  Path file1snap1 = SnapshotTestHelper.
      getSnapshotPath(sub, snapshot1, file1Name);
  FSDataOutputStream out = hdfs.append(file1);
  AppendTestUtil.write(out, 0, toAppend);
  byte[] dataFromSnapshot = DFSTestUtil.
      readFileBuffer(hdfs, file1snap1);
  assertThat("Wrong data size in snapshot.",
      dataFromSnapshot.length, is(origLen));
}
```

Test for HDFS snapshots



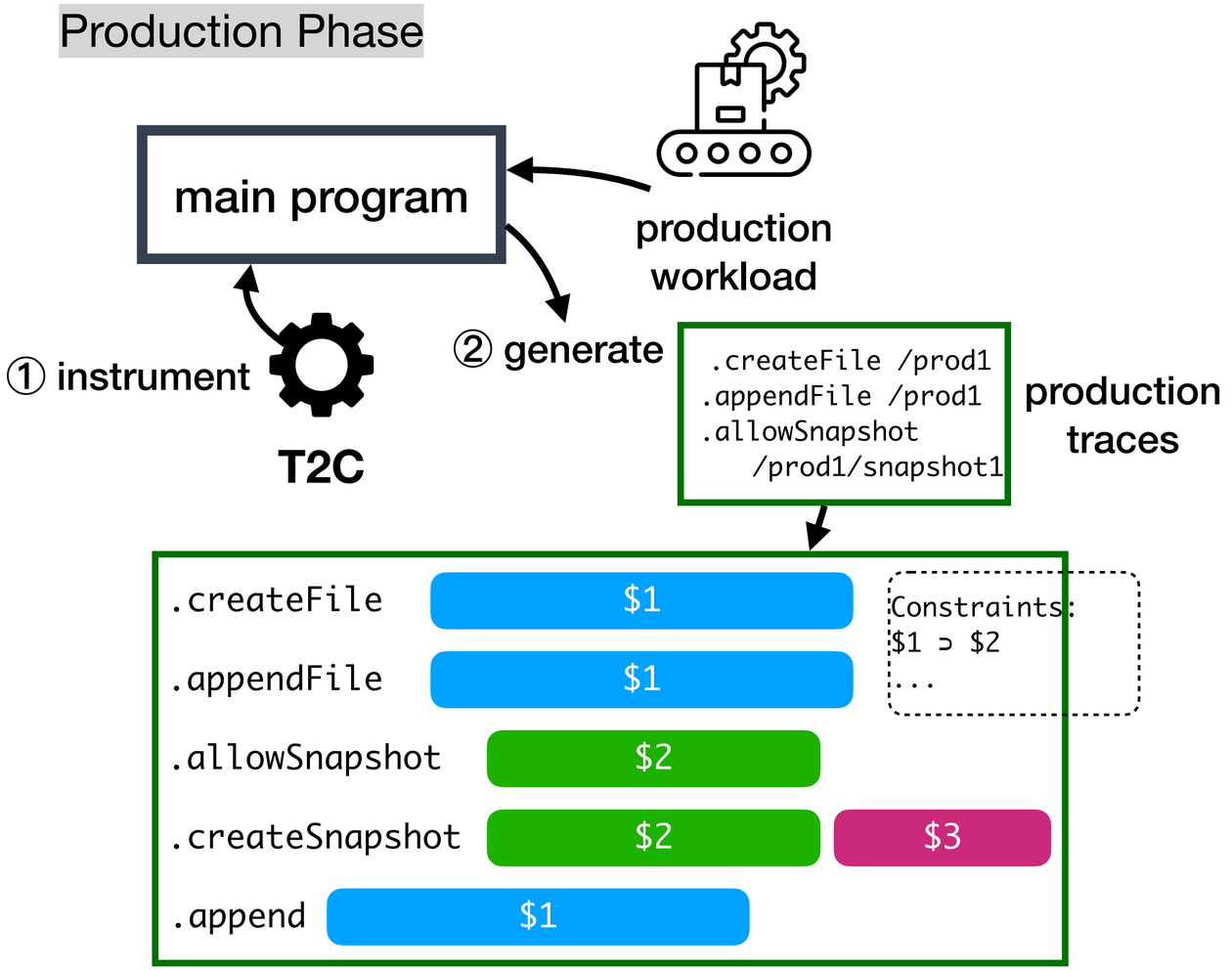
Precondition

#2 Identify checker precondition with symbolization

```
public void testSnapshotfileLength() {
  Path file1 = new Path(sub, file1Name);
  DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
        0, BLOCKSIZE, REPLICATION, SEED);
  DFSTestUtil.appendFile(hdfs, file1, origLen);
  hdfs.allowSnapshot(sub);
  hdfs.createSnapshot(sub, snapshot1);

  Path file1snap1 = SnapshotTestHelper.
      getSnapshotPath(sub, snapshot1, file1Name);
  FSDataOutputStream out = hdfs.append(file1);
  AppendTestUtil.write(out, 0, toAppend);
  byte[] dataFromSnapshot = DFSTestUtil.
      readFileBuffer(hdfs, file1snap1);
  assertThat("Wrong data size in snapshot.",
      dataFromSnapshot.length, is(origLen));
}
```

Test for HDFS snapshots



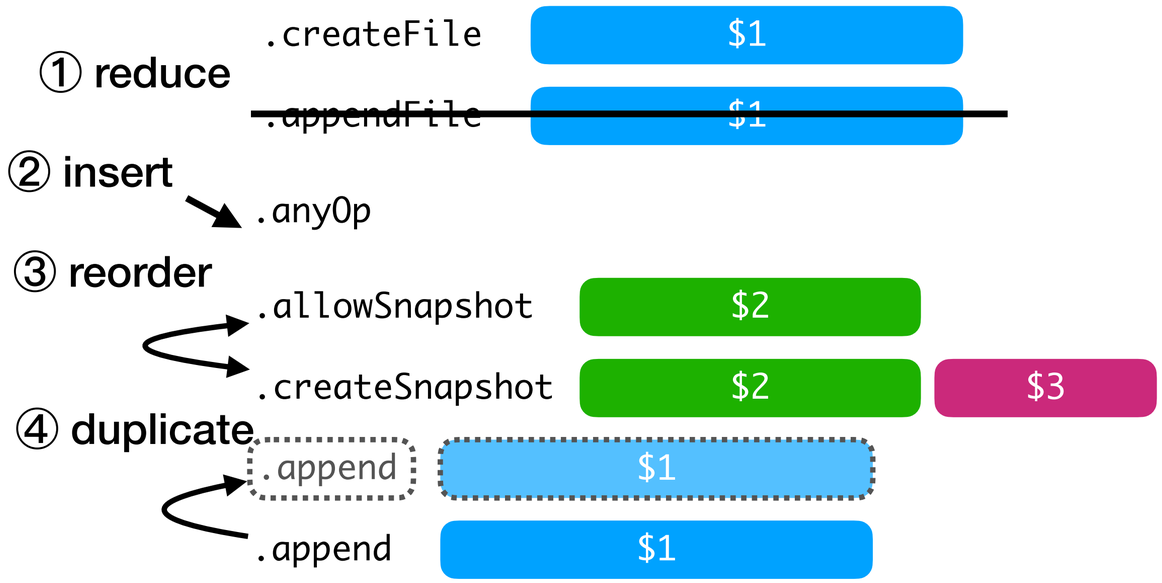
Precondition

#3 Generalize checker precondition with mutation

```
public void testSnapshotfileLength() {
  Path file1 = new Path(sub, file1Name);
  DFSTestUtil.createFile(hdfs, file1, BLOCKSIZE,
        0, BLOCKSIZE, REPLICATION, SEED);
  DFSTestUtil.appendFile(hdfs, file1, origLen);
  hdfs.allowSnapshot(sub);
  hdfs.createSnapshot(sub, snapshot1);

  Path file1snap1 = SnapshotTestHelper.
      getSnapshotPath(sub, snapshot1, file1Name);
  FSDataOutputStream out = hdfs.append(file1);
  AppendTestUtil.write(out, 0, toAppend);
  byte[] dataFromSnapshot = DFSTestUtil.
      readFileBuffer(hdfs, file1snap1);
  assertThat("Wrong data size in snapshot.",
      dataFromSnapshot.length, is(origLen));
}
```

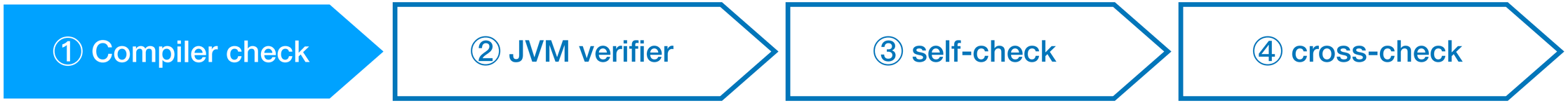
Test for HDFS snapshots



Precondition (mutated)

Checker validation

– T2C uses four levels of validations to prune invalid checkers



```
error: incompatible types: String
cannot be converted to int
```

Checker validation

– T2C uses four levels of validations to prune invalid checkers

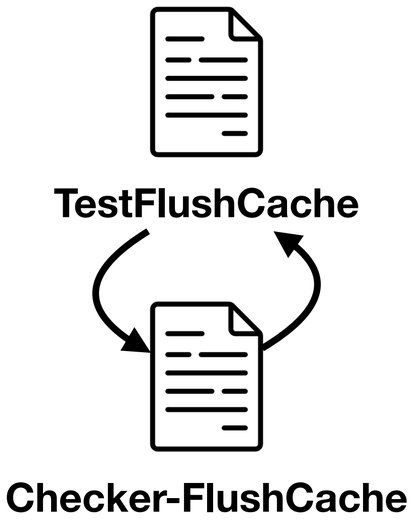
① Compiler check → ② JVM verifier → ③ self-check → ④ cross-check

```
Exception in thread "main" java.lang.VerifyError:  Mismatched
stack types
    at java.lang.Class.getDeclaredConstructors0(Native Method)
    at
java.lang.Class.privateGetDeclaredConstructors(Class.java:2446)
    at java.lang.Class.getConstructor0(Class.java:2756)
    at java.lang.Class.newInstance0(Class.java:328)
    at java.lang.Class.newInstance(Class.java:310)
```

Checker validation

– T2C uses four levels of validations to prune invalid checkers

① Compiler check → ② JVM verifier → ③ self-check → ④ cross-check

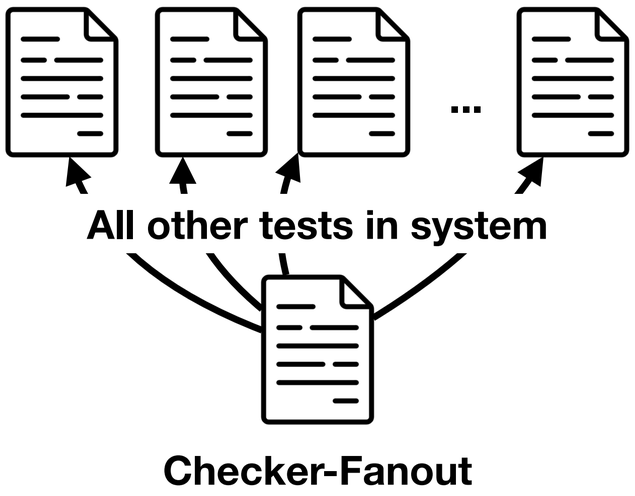



```
java.lang.AssertionError: toggle=false i=940 ts=1484852861597
expected:<94> but was:<92>
  at org.junit.Assert.fail(Assert.java:88)
  at org.junit.Assert.failNotEquals(Assert.java:834)
  at org.junit.Assert.assertEquals(Assert.java:645)
  at
org.apache.hadoop.hbase.regionserver.TestHRegion.testFlushCacheW
hileScanning(TestHRegion.java:3533)
```

Checker validation

– T2C uses four levels of validations to prune invalid checkers

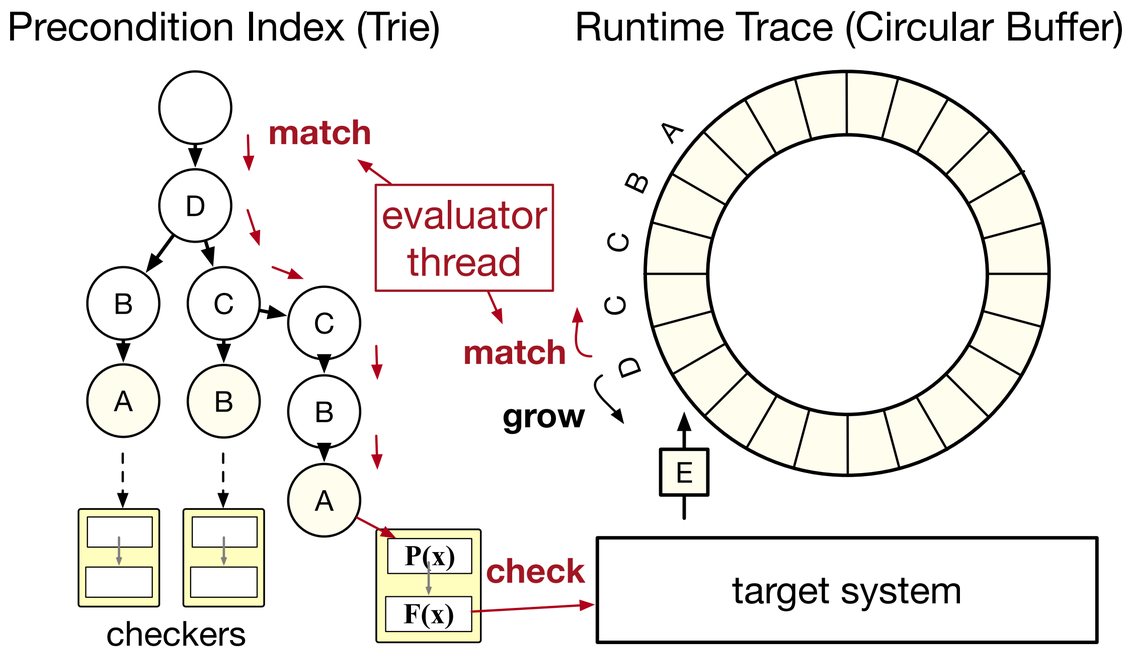
① Compiler check → ② JVM verifier → ③ self-check → ④ cross-check




```
java.lang.AssertionError: expected:<2> but was:<1>
        at org.junit.Assert.fail(Assert.java:89)
        at org.junit.Assert.failNotEquals(Assert.java:835)
        at org.junit.Assert.assertEquals(Assert.java:647)
        at org.junit.Assert.assertEquals(Assert.java:633)
        at
org.apache.hadoop.hbase.io.asyncfs.TestFanOutOneBlockAsyncDFSOut
put.testConnectToDatanodeFailed(TestFanOutOneBlockAsyncDFSOutput
.java:220)
```

Checker deployment

- T2C deploy checkers together with the target system to production
  - performance **bottleneck**: (i) bookkeeping and (ii) matching traces
  - solution: optimized data structure



Evaluation

Evaluation: checker generation

Software	Version	Classes	Tests (total)	Tests (target)	Tests (transformed)	Checker (healthy)	Checker (validated)
ZooKeeper	3.4.11	123	428	109	100	90	46
Cassandra	3.11.5	604	3283	257	242	232	100
HDFS	3.2.2	803	4304	816	729	707	230
HBase	2.4	980	4287	990	948	904	296

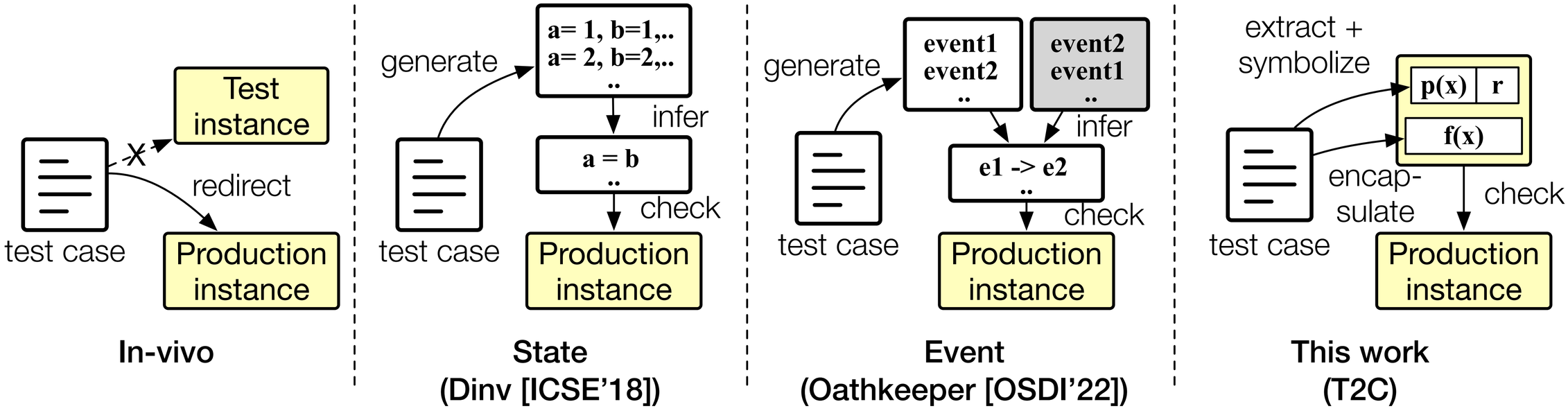
Evaluation: detection

JIRA Id.	Id.	Sys. Feature	Description
ZooKeeper-2355	ZK1	Ephemeral Node	Wrong Results
ZooKeeper-1208	ZK2	Ephemeral Node	Wrong Results
ZooKeeper-1754	ZK3	Read-only Mode	Illegal Operation
ZooKeeper-4026	ZK4	Multi Request	Inconsistency
ZooKeeper-4362	ZK5	Transaction	Inconsistency
ZooKeeper-4325	ZK6	Deletion	Data Corruption
ZooKeeper-4646	ZK7	Transaction	Data Loss
Cassandra-15072	CS1	Range Query	Wrong Results
Cassandra-14873	CS2	Static Row	Wrong Results
Cassandra-14092	CS3	Time-to-live Data	Data Loss
Cassandra-14803	CS4	Read Query	Wrong Results
HDFS-14514	HF1	Snapshot	Broken Redundancy
HDFS-14699	HF2	Erasure Coding	Data Loss
HDFS-16633	HF3	Storage Management	Redundant Workloads
HDFS-16942	HF4	Replication	Broken Redundancy
HBase-21041	HB1	Memstore	Incorrect State
HBase-25827	HB4	Time-to-live Data	Data Corruption
HBase-21612	HB5	Reversed Scan	Wrong Results

20 real-world silent failures reproduced for evaluation

Evaluation: detection

— We compare our approach with three baselines



Evaluation: detection

	ZK1	ZK2	ZK3	ZK4	ZK5	ZK6	ZK7	CS1	CS2	CS3	CS4	HF1	HF2	HF3	HF4	HB1	HB2	HB3	HB4	HB5
In-vivo	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	37.05	✗	✗	✗	✗	✗
State	1.154	1.183	✗	✗	0.561	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Event	✗	✗	✗	✗	✗	3.433	3.569	✗	✗	✗	✗	✗	3.982	4.783	3.221	✗	✗	✗	✗	✗
T2C	0.112	1.018	0.460	✗	0.311	✗	✗	0.177	0.035	0.304	0.029	0.570	2.459	✗	0.039	1.715	✗	0.188	0.041	0.021

Detection times (in secs) for the real-world cases

T2C detects 15 out of 20 evaluated failure cases

New bug exposed in the latest ZooKeeper

ZooKeeper / ZOOKEEPER-4837

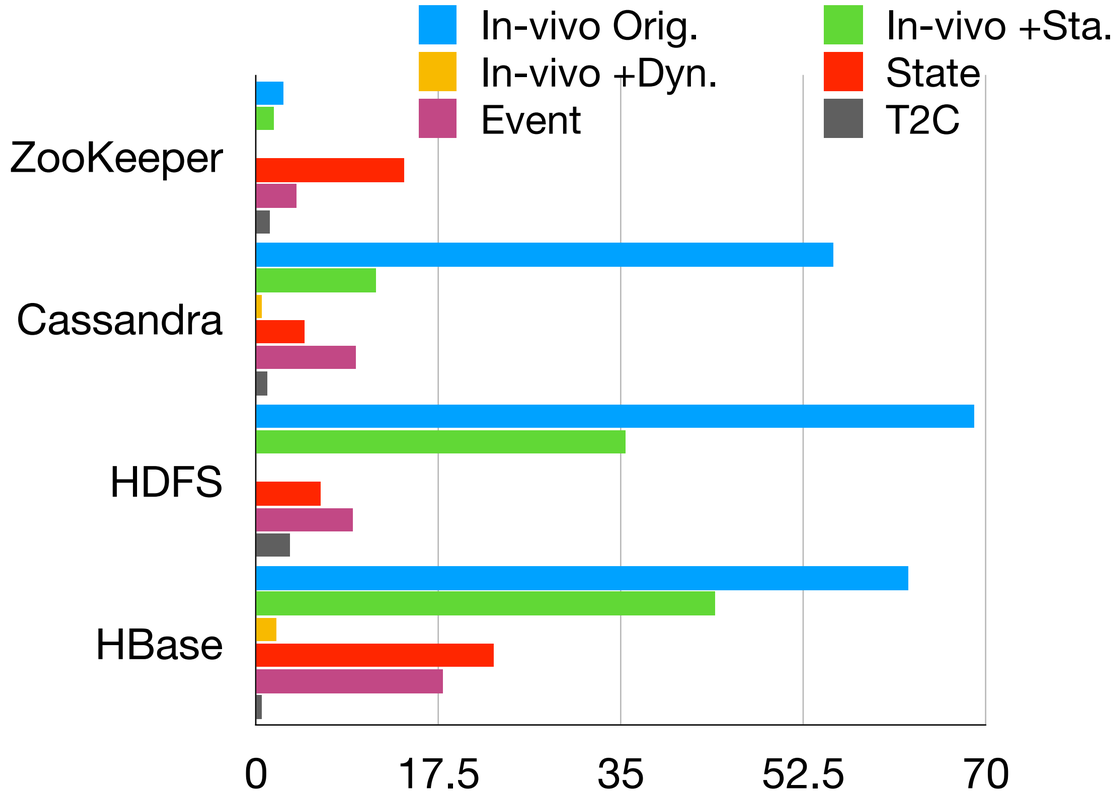
Network issue causes ephemeral node unremoved after the session expiration

In our testing cluster with the latest ZooKeeper version (66202cb), we observed that sometimes an ephemeral node never gets deleted if there is a network issue during the PROPOSAL request, even after the session expires. This bug is essentially related to ~~ZOOKEEPER-2355~~, but the issue was not entirely fixed in the previous patch. We also tested on some related open PRs (e.g., https://github.com/apache/zookeeper/pull/2152 and https://github.com/apache/zookeeper/pull/1925), and confirmed the issue exists after the proposed fix.

Steps to reproduce this bug:

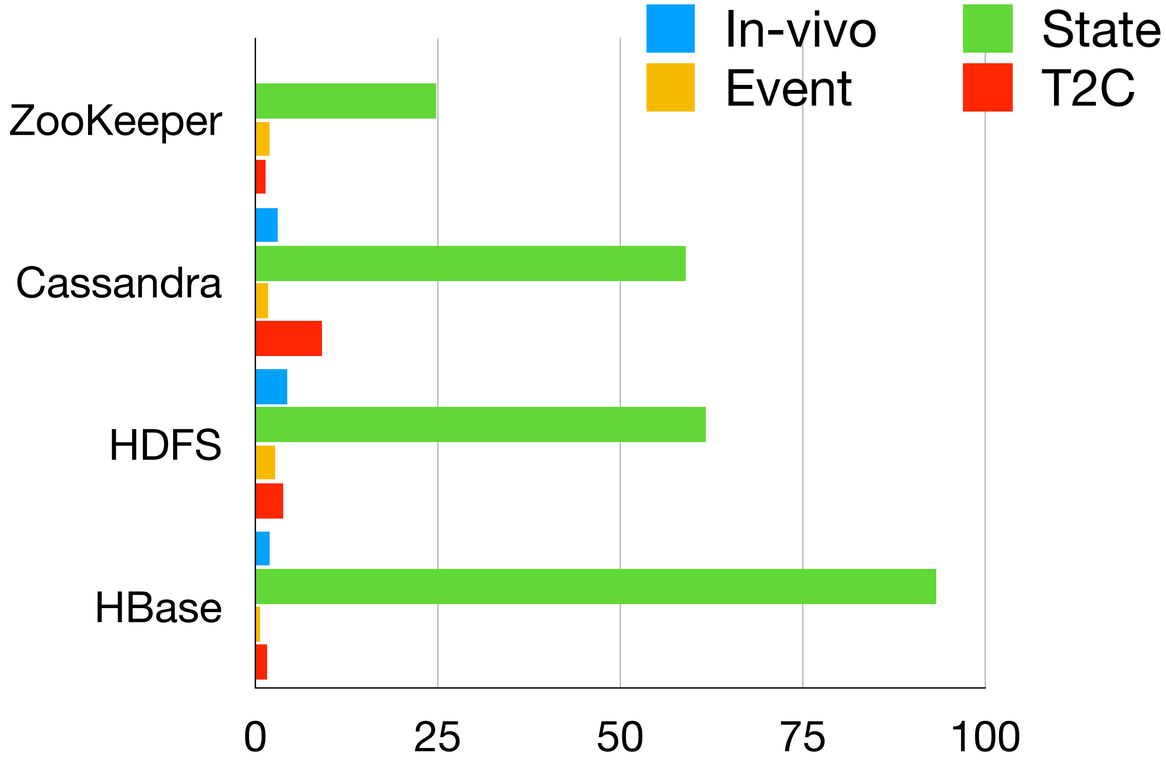
1. Start a cluster with 3 servers, follower A, leader B, follower C
2. Open a zk client in server A
3. Create an ephemeral node in the client
4. Inject network issue in all server that causes SocketTimeoutException during readPacket if the packet is a PROPOSAL
5. Close the client
6. Wait until the cluster is stable (the leader will change between B and C several times)
7. Remove the network issue from all server
8. Check every server for ephemeral node existence. The ephemeral node will exist in server A. However, server B and C will not have the ephemeral node anymore.

Evaluation: false alarms and overhead



T2C generates low false positives (1%-3%)

Key factor: validation

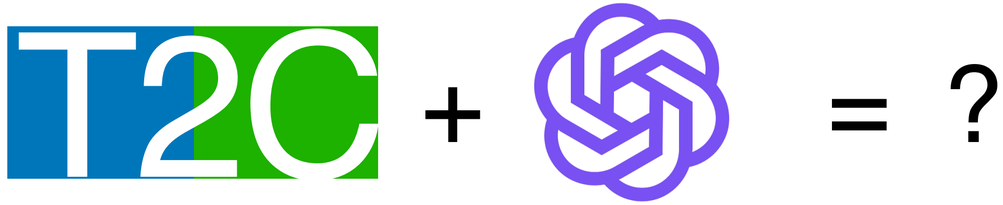


T2C generates a 4.0% runtime overhead on avg

Key factors: precise precondition && optimized structure

Conclusion

- Silent (semantic) failures are a severe threat to cloud system availability
- Test cases contain valuable information about the system semantics
- T2C derives semantic checkers from tests to detect silent failures
  - evaluated on ZooKeeper, Cassandra, HDFS and HBase
  - generated hundreds of validated checkers
  - detected real-world silent failures with small overhead



github.com/OrderLab/T2C

chlou@virginia.edu