Demystifying and Checking Silent Semantic Violations in Large Distributed Systems

Chang Lou, Yuzhuo Jing, Peng Huang
Distributed systems provide rich semantics

Client APIs
- watch, kill, prune, reconnect..

Component guarantees
- message ordering, redundancy, ACID..

Configurations
- tickTime, snapCount, maxClientCnxns..
Semantics encode various promises

- **promise 1:** exactly-once
- **promise 2:** n-way redundancy
- **promise 3:** watch triggers when updated

Client 1

Client 2

DATA_CHANGE_EVENT

Replicas
When a promise is not a promise

Client 1

promise 1: exactly-once

Client 2

promise 3: watch triggers when updated

promise 2: n-way redundancy

Replicas

REPLICA MISSING
Semantic violations

- **Promise 1:** exactly-once
- **Promise 2:** n-way redundancy
- **Promise 3:** watch triggers when updated

Client 1

Client 2
Existing work focus on failures w/ explicit errors.
Silent semantic violations

Client 1

![Client 1](image1)

Client 2

![Client 2](image2)

- REQ_SUCCESS
- (actually failed to process req)
- length < 0?
- corrupt state
- data loss
- perf. degrade
- security issue

Replicas

![Replicas](image3)
Contributions

1. A study on \textbf{109} real-world silent semantic violations
   - cases collected from 9 popular distributed systems

2. A detection solution: Oathkeeper
   - automatically \textit{infer} semantic rules from \textit{past failures}
   - enforce the rules at runtime to \textit{detect} new failures
Study methodology

- Study on real-world incidents from nine distributed systems
  - randomly sampled 747 user-reported failures in total
  - confirmed 268 cases as silent semantic violations
  - performed in-depth studies on 109 cases

<table>
<thead>
<tr>
<th>System</th>
<th>Category</th>
<th>Lang.</th>
<th>Total</th>
<th>Sampled Confirmed</th>
<th>Studied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra</td>
<td>Database</td>
<td>Java</td>
<td>54</td>
<td>25</td>
<td>12</td>
</tr>
<tr>
<td>CephFS</td>
<td>File System</td>
<td>C++</td>
<td>123</td>
<td>37</td>
<td>12</td>
</tr>
<tr>
<td>ElasticSearch</td>
<td>Search</td>
<td>Java</td>
<td>46</td>
<td>26</td>
<td>10</td>
</tr>
<tr>
<td>HBase</td>
<td>Database</td>
<td>Java</td>
<td>80</td>
<td>32</td>
<td>14</td>
</tr>
<tr>
<td>HDFS</td>
<td>File System</td>
<td>Java</td>
<td>52</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>Kafka</td>
<td>Streaming</td>
<td>Scala</td>
<td>92</td>
<td>39</td>
<td>13</td>
</tr>
<tr>
<td>Mesos</td>
<td>Cluster Manager</td>
<td>C++</td>
<td>47</td>
<td>21</td>
<td>12</td>
</tr>
<tr>
<td>MongoDB</td>
<td>Database</td>
<td>C++</td>
<td>151</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>ZooKeeper</td>
<td>Coordination</td>
<td>Java</td>
<td>102</td>
<td>36</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>747</td>
<td>268</td>
<td>109</td>
</tr>
</tbody>
</table>
Major findings

- [Prevalence] How common are silent semantic violations in production?
- [Age of semantics] How long has the violated semantics existed?
- [Testing] Is semantics covered by tests and why did not expose issue?
- [Root cause] Can we find common bug patterns for static checking?
- [Timing] When do semantic violations happen?
- ...

Prevalence

- Myth: are silent semantic violations rare in production?
• Myth: are silent semantic violations rare in production?

• Finding 1: silent semantic violations are prevalent
  • occupy 39% of cases for all types of failures

```
... invariant(!msg->empty());
invariant(msg->operation() == dbMsg);
invariant(msg->dataSize() >= sizeof(uint32_t));
DataView(msg->data()).write(flags);
...
```

MongoDB has lowest ratio
Age of semantics

- Myth: violated semantics are fragile because they are new?
Myth: violated semantics are fragile because they are new?

Finding 2: 68% of the studied failures violate old semantics
- "old" means semantics exist since the first major release of the system
- same semantics is repeatedly violated, e.g., ZooKeeper ephemeral node

Age of semantics
Age of semantics

• Myth: violated semantics are fragile because they are new?

• Finding 2: 68% of the studied failures violate old semantics
  • "old" means semantics exist since the first major release of the system
  • same semantics is repeatedly violated, e.g., ZooKeeper ephemeral node

create('/node1')

violation: node1 not deleted!

session start session end

ephemeral node exists since first major release

2008

ZK-1208 2011

ZK-2355 2016

... 2022

46 related failures

ZK-4541
Testing

- Myth: does violated semantics have poor testing?
Myth: does violated semantics have poor testing?

Finding 3: 73% of violated semantics are covered by existing tests

- lack operations, arguments, timing to expose new violations
- existing efforts of writing tests do not effectively prevent future violations

Violation: snapshot has viable size

Appending File + Create Snapshot

- triggering conditions in existing test
- triggering conditions for new violation

Append File
Create Snapshot
Triggering conditions in existing test

Snapshot capture open files = true
Dfs Admin create Encryption Zone

Close + Append File + Create Snapshot

Triggering conditions for new violation
Root causes

- Myth: do same semantic violations have similar causes?
Myth: do same semantic violations have similar causes?

Finding 4: root causes are diverse
• even for failures violating the same semantics, the causes are often different

- create ephemeral node
- close session
- node should be removed

semantics: ephemeral node
Root causes

- Myth: do same semantic violations have similar causes?

- Finding 4: root causes are diverse
  - even for failures violating the same semantics, the causes are often different

![Diagram of root causes]

- ZK-1208: race condition
Root causes

- Myth: do same semantic violations have similar causes?

- Finding 4: root causes are **diverse**
  - even for failures violating the same semantics, the causes are often different

```plaintext
create ephemeral node
  ▼
  close session
  ▼
node should be removed
```

```plaintext
create ephemeral node
  ▼
transient network fault
  ▼
syncWithLeader
  ▼
node not removed

semantics: ephemeral node
```

2 ZK-2355: buggy error handling
Root causes

- Myth: do same semantic violations have similar causes?

- Finding 4: root causes are diverse
  - even for failures violating the same semantics, the causes are often different

semantics: ephemeral node
Timing of violation

- Myth: appending a check after each operation can solve problem
Myth: appending a check after each operation can solve problem

Finding 5: 67% of cases violate long-lived semantics
Finding 5: 67% of cases violate long-lived semantics

- Myth: appending a check after each operation can solve problem

- Example of short-lived semantics (33%):

  ```sql
  > UPDATE cyclists SET name = 'Alex' WHERE id = 11;
  > SELECT name FROM cyclists WHERE id = 11;
  Alice
  ```
Timing of violation

- Myth: appending a check after each operation can solve problem

- Finding 5: 67% of cases violate long-lived semantics

![Diagram showing timing of violation]

- Short-lived semantics (33%)
  - (10%)

- Long-lived semantics (67%)
  - (40%)
  - (17%)
Timing of violation

- Myth: appending a check after each operation can solve problem
- Finding 5: 67% of cases violate long-lived semantics

HBASE-20588

\[
\text{\begin{tabular}{l}
\text{op1\_start} & \text{op1\_end} \\
\hline
\end{tabular}}
\]

\[
\text{\begin{tabular}{l}
\text{op2\_start} & \text{op2\_end} \\
\hline
\end{tabular}}
\]

- short-lived semantics (33%)
- long-lived semantics (67%)

\[
\text{\begin{tabular}{l}
\text{X} & \text{X} & \text{X} & \text{X} \\
\hline
\end{tabular}}
\]

\[
\text{\begin{tabular}{l}
\text{(10%)} & \text{(40%)} & \text{(17%)} \\
\hline
\end{tabular}}
\]

\[
\text{\begin{tabular}{l}
\text{HBASE-20588} \\
\end{tabular}}
\]

\[
\text{\begin{tabular}{l}
\text{\begin{tabular}{l}
\text{\textbackslash{}> \text{set\_quota} TYPE => SPACE, TABLE => 'TestTable',} \\
\text{\textbackslash{}LIMIT => '2M', POLICY => NO\_INSERTS} \\
\text{\textbackslash{}> put 'TestTable', '1'} \\
\text{\textbackslash{}ERROR:} \\
\text{\textbackslash{}org.apache.hadoop.hbase.quotas.SpaceLimitingException} \\
\end{tabular}}
\end{tabular}}
\]

\[
\text{\begin{tabular}{l}
\text{\begin{tabular}{l}
\end{tabular}}
\end{tabular}}
\]
Timing of violation

- Myth: appending a check after each operation can solve problem

- Finding 5: 67% of cases violate long-lived semantics
Myth: appending a check after each operation can solve problem

Finding 5: 67% of cases violate long-lived semantics

- create ephemeral node
- close session
- transient network fault
- check ephemeral node

still exists

short-lived semantics (33%)

long-lived semantics (67%)

op1_start

op2_start  op2_end

ZK-2355

(10%)  (40%)  (17%)
Other findings

- Finding 6: sanity checks are insufficient
  - in 51% of the failures the buggy functions have some sanity checks
  - only 9% cases can be potentially detected by adding proper sanity checks

- Finding 7: local vs. distributed semantics

- Finding 8: safety vs. liveness semantics

- Finding 9: user observability

- ...

See the full list of findings in our paper
Oathkeeper: a semantic violation detection tool

- **Motivating findings:**
  - the majority of studied failures violate old semantics
  - the testing coverage of these semantics is decent
  - the same semantics is repeatedly violated by different root causes
  - many failures violate long-lived semantics
  ...
Oathkeeper: a semantic violation detection tool

- Motivating findings:
  - the majority of studied failures violate old semantics
  - the testing coverage of these semantics is decent
  - the same semantics is repeatedly violated by different root causes
  - many failures violate long-lived semantics

- Key idea:
  - extract essence from semantic failure regression tests and enforce it
Oathkeeper workflow

1. Instrument-lib
2. Regression test
3. Inference engine
4. Semantic rules
5. Verification
6. Violation alert

Pre-production
Production

W/o patch
W/ patch

W/o patch
W/ patch

Buggy trace
Patched trace

Semantic rules

Production system

Instrument-lib
How to express semantics?

Relationship among semantics-related events
(obtained from instrumentation)

1 public void serialize(...) {
2    logEvent(Type.Op, "serialize", ...);
...

1 Map<Long, HashSet<String>> ephemerals;
...
71 void killSession(long session, long zxid) {
72    HashSet<String> list = ephemerals.remove(session);
73    logEvent(Type.State, "ephemerals", "killSession",
74    ephemerals, ...);
...

Predicates over key state variables:

Dinv¹

0 <= Sender <= N
∀ nodes i, j, NodeStateᵢ = NodeStateⱼ

D3S²

∀ l ∈ LockID, sizeof(Owners(l)) <= 1

Emitting semantic event traces

```java
void testCreateAfterCloseShouldFail() {
    // open a connection
    ConnectRequest conReq = new ...
    // close connection
    RequestHeader h = new ...
    // create ephemeral znode (race)
    CreateRequest createReq = new ...
    assertEquals(1, zk.getChildren("/").size()); }
```

regression test (ZK-1208)
General semantic rule templates

- Relation examples summarized from study

<table>
<thead>
<tr>
<th>Template</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p \Rightarrow q)</td>
<td>decommission a datanode should trigger reconstruction</td>
</tr>
<tr>
<td>(s \uparrow \Rightarrow p)</td>
<td>when datanode changes, associated watcher notifies clients</td>
</tr>
<tr>
<td>(s \uparrow \Rightarrow k \uparrow)</td>
<td>after session disconnection, ephemeral node is removed</td>
</tr>
<tr>
<td>((s = c) \oplus q)</td>
<td>deny new requests after connections reach maxClientCnxns</td>
</tr>
<tr>
<td>(p + \Delta t \Rightarrow q)</td>
<td>read-only server should not provide write access</td>
</tr>
<tr>
<td>(s \uparrow \rightarrow q)</td>
<td>inserted data should expire after the TTL is reached.</td>
</tr>
<tr>
<td>(p \Rightarrow \ominus(s \uparrow, k \uparrow))</td>
<td>after snapshot renaming, either new snapshot creation and old snapshot deletion both</td>
</tr>
</tbody>
</table>

full template list included in our tech report
Inference example: \( p \Rightarrow q \)

- Assume all rules hold and filter rules if counterexamples found

<table>
<thead>
<tr>
<th>input</th>
<th>([e1, e2, e3, e1, e2])</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-scan</td>
<td></td>
</tr>
<tr>
<td>(&lt;e1,e2&gt;)</td>
<td>(&lt;e2,e1&gt;)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e1</td>
<td>1</td>
</tr>
<tr>
<td>e2</td>
<td>0</td>
</tr>
<tr>
<td>scan</td>
<td></td>
</tr>
<tr>
<td>e3</td>
<td>0</td>
</tr>
<tr>
<td>e1</td>
<td>1</td>
</tr>
<tr>
<td>e2</td>
<td>0</td>
</tr>
<tr>
<td>post-scan</td>
<td></td>
</tr>
<tr>
<td>e1 (\Rightarrow e2)</td>
<td>e3 (\Rightarrow e1)</td>
</tr>
</tbody>
</table>
Validation example: $p \Rightarrow q$

- Only preserve rules that are violated in buggy trace

<table>
<thead>
<tr>
<th>input</th>
<th>[e1, e2, e3, e1]</th>
<th>e1 ⇒ e2</th>
<th>e3 ⇒ e1</th>
<th>e3 ⇒ e2</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-scan</td>
<td>e1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>e2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>e3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>e1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>scan</td>
<td>e1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>e2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>e3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>post-scan</td>
<td>e1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Validation against all tests

- False rules may still remain after validating against buggy trace
- The verifier further validates rules against traces from all tests
  - mark rules without counterexamples as verified
Runtime detection

- In production, the target system is deployed with verifier and instrumentation library
- Only rule-related functions are instrumented
- Deployed semantic rules periodically validate against the runtime trace
  - report alerts in the log with counterexamples
Runtime detection

- In production, the target system is deployed with verifier and instrumentation library

```java
[...]
ASSERT FAIL! #220
Invariant{template=oathkeeper.runtime.template.StateUpdateImplyStateUpdateTemplate,
            context=Context{
                left=StateUpdateEvent{state='org.apache.zookeeper.server.DataTree.ephemerals'..},
                right=StateUpdateEvent{state='org.apache.zookeeper.server.SessionTracker.sessionsById'..}
            }
Conflicts with trace: [
    ...
```

- report alerts in the log with counterexamples
Optimizations

‣ "Survivor" mode for validation
  • prioritize running related tests to invalidate rules more efficiently
  • reduce validation processing time

‣ Ring buffer tracer for runtime
  • reuse expired event objects
  • effectively lower runtime overhead
Evaluation

‣ Integrated Oathkeeper with ZooKeeper, HDFS and Kafka

‣ We try to answer questions such as
  • can Oathkeeper check new violations from past failures?
  • is runtime checking accurate?
  • how fast can tool generate rules?
  • is runtime checking lightweight?
We select old semantic failures and regression tests to reproduce:

- extracted 285 rules for ZooKeeper, 1,209 rules for HDFS, and 150 rules for Kafka

<table>
<thead>
<tr>
<th>ZooKeeper</th>
<th>HDFS</th>
<th>Kafka</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZK-1046</td>
<td>HDFS-8950</td>
<td>KAFKA-9144</td>
</tr>
<tr>
<td>ZK-1208</td>
<td>HDFS-9204</td>
<td>KAFKA-9491</td>
</tr>
<tr>
<td>ZK-1412</td>
<td>HDFS-10192</td>
<td>KAFKA-9666</td>
</tr>
<tr>
<td>ZK-1573</td>
<td>HDFS-10536</td>
<td>KAFKA-9752</td>
</tr>
<tr>
<td>ZK-1754</td>
<td>HDFS-10968</td>
<td>KAFKA-9891</td>
</tr>
<tr>
<td>ZK-1755</td>
<td>HDFS-11960</td>
<td>KAFKA-9921</td>
</tr>
<tr>
<td>ZK-2680</td>
<td>HDFS-12862</td>
<td>KAFKA-10001</td>
</tr>
<tr>
<td>ZK-2797</td>
<td>HDFS-13120</td>
<td>KAFKA-10545</td>
</tr>
<tr>
<td></td>
<td>HDFS-13192</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HDFS-14504</td>
<td></td>
</tr>
</tbody>
</table>
Detecting real-world failures

- Oathkeeper detects violations for 6 of 7 evaluated cases
  - use regression tests 9–34 months earlier than new failures
  - baseline checker based on Dinv\(^1\) only detects 1 case

<table>
<thead>
<tr>
<th>JIRA Id</th>
<th>Violated Semantics</th>
<th>Rules from</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZK-1496</td>
<td>ephemeral node should be deleted after session expired</td>
<td>ZK-1208</td>
</tr>
<tr>
<td>ZK-1667</td>
<td>watcher should return correct event when client reconnected</td>
<td>MISS</td>
</tr>
<tr>
<td>ZK-3546</td>
<td>container node should be deleted after children all removed</td>
<td>ZK-2705</td>
</tr>
<tr>
<td>HDFS-14699</td>
<td>failed block need to be reconstructed</td>
<td>HDFS-10968</td>
</tr>
<tr>
<td>HDFS-14317</td>
<td>edit log rolling should be activated periodically</td>
<td>HDFS-10536</td>
</tr>
<tr>
<td>HDFS-14633</td>
<td>file rename should respect storageType quota</td>
<td>HDFS-14504</td>
</tr>
<tr>
<td>KAFKA-12426</td>
<td>partition topic ID should be persisted into metadata file</td>
<td>KAFKA-10545</td>
</tr>
</tbody>
</table>

Generated rules incur 4-12% false positive ratios

- greatly benefits from the validation steps
- can be further reduced by adding profile runs or a dynamic ban mechanism
Offline performance

- Trace generation and inference usually take up to minutes

- Validation is most time-consuming part
  - survivor mode can reduce validation time by 38%

<table>
<thead>
<tr>
<th>Phase</th>
<th>Median time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace generation</td>
<td>153.5</td>
</tr>
<tr>
<td>inference</td>
<td>6.5</td>
</tr>
<tr>
<td>validation</td>
<td>2,196</td>
</tr>
</tbody>
</table>
Oathkeeper adds ~1.27% overhead on throughput

- overhead is mainly from the added instrumentation to emit traces
- ring buffer optimization eliminates overhead by frequent GC
Conclusion

- Semantics in distributed systems can be violated silently
- Our study reveals interesting findings
  - same old semantics can be violated repeatedly in different scenarios
  - long-lived semantics require continuous monitoring
- Oathkeeper: a runtime detection tool
  - infer semantic rules from past failures to detect new violations

https://github.com/OrderLab/OathKeeper