Automatic Reliability Testing for Cluster Management Controllers

Xudong Sun, Wenqing Luo, Jiawei Tyler Gu, Aishwarya Ganesan, Ramnatthan Alagappan, Michael Gasch, Lalith Suresh, Tianyin Xu
Kubernetes: Manage applications, services, networking, storage, etc. Cluster management is realized by controllers. Thousands of controllers being developed by the Kubernetes community.
Cluster management is realized by controllers

Twitter joins the rest of the world – moves to Kubernetes

Thousands of controllers being developed by the Kubernetes community
Controllers implement state reconciliation

Objects representing containers, volumes, nodes, applications...

Goal: Reconcile the current state to the desired state
Controllers implement state reconciliation

Container
Volume

Current

Cassandra Controller
Delete(container)
Delete(volume)

Desired

Controllers implement state reconciliation
Controller reliability is critical, but challenging!

Volume
- Controller malfunction
- Resource leak
- Security issue
- Crash
- and restart

Current

Desired

Cassandra Controller

Delete(container)

Delete(volume)

Never executed

This bug was detected by our tool and has been fixed by the developers.
Contributions

• Sieve: automatic reliability testing for Kubernetes controllers
  • Key idea: Perturbing the controller’s view of cluster state
  • Usability: Testing unmodified controllers
  • Reproducibility: Reproducing detected bugs reliably
  • Open sourced at https://github.com/sieve-project/sieve

• Detected 46 serious bugs in 10 popular Kubernetes controllers
  • Severe consequences: System outage, data loss, security issues, etc.
  • 35 confirmed and 22 fixed
Challenges of testing controllers

- Controller malfunction
- Resource leak
- Security issue

Non-crashing symptom
Sophisticated triggering condition

Different implementations and diverse functionality

- Controller malfunction
- Resource leak
- Security issue
Perturb the controller’s view of cluster state

Reference run

Initial state

Desired state

Cluster state: Objects in etcd
Every object creation/update/deletion advances the state

Perturbed run

A controller makes reconciliation decisions based on its view of the current cluster state.

Common, transient faults
Flag buggy behavior with **differential oracles**

Reference run

Initial state

Desired state

Cluster state: Objects in **etcd**

Perturbed run

Differential oracles:
Detecting liveness and safety violations without knowing the semantic of the state objects

Common, transient faults
Flag buggy behavior with differential oracles

Reference run

Initial state

Cluster state: Objects in etcd

Desired state

Perturbed run

Liveness Property
A controller should eventually achieve the desired state

Compare the end states

Common, transient faults
Flag buggy behavior with *differential oracles*

**Reference run**

Initial state

Desired state

**Cluster state: Objects in `etcd`**

**Safety Property**

A controller should *never* delete user data unless requested

**Compare the state updates (e.g., # volume deletions)**

**Perturbed run**

Common, transient faults
Interposition around state-centric interface

- State-centric interface is used to read/write state objects
- Automatic interposition around cluster state transitions
- Allow Sieve to test *unmodified* controllers
Detect diverse controller bugs

• Employ **three perturbation patterns**

• **Exhaustively** test all bug-triggering perturbations
  • Systematically find all the targeted bugs
  • Inject faults with different timings

• Prune out ineffective perturbations to be **efficient**
  • Not every perturbation leads to bugs
The intermediate-state pattern

Reference run

\[
\begin{align*}
S1 \\
S2 \\
S3 \\
S4 \\
S5
\end{align*}
\]

Intermediate state

Perturbed run

\[
\begin{align*}
S1 \\
S2
\end{align*}
\]

Intermediate state

No atomicity guarantee!

\[
\begin{align*}
\{ \text{Create(...) //S1->S2} \\
\text{Update(...) //S2->S3}
\end{align*}
\]

\text{Crash}

A controller should correctly handle any intermediate state.

Start a new reconcile cycle from S2
An intermediate-state bug detected by Sieve

Reference run

```java
switch (phase) {
    case "Ongoing":
        if (ContainerNotFound(container)) {
            return Error("Container not found")
        }
        ...
        Delete(container)
        ...
        UpdatePhase("Finalizing") // phase <- "Finalizing"
        ...
        case "Finalizing":
        ...
        Delete(volume)
        ...
        UpdatePhase("Done") // phase <- "Done"
    }
// https://github.com/Orange-OpenSource/casskop
```
switch phase {
    case "Ongoing":
        if ContainerNotFound(container) {
            return Error("Container not found")
        }
        ...
        Delete(container)
        ...
        UpdatePhase("Finalizing") // phase <- "Finalizing"
        ...
    case "Finalizing":
        ...
        Delete(volume)
        ...
        UpdatePhase("Done") // phase <- "Done"
    }
    // https://github.com/Orange-OpenSource/casskop

Always returns here...

Never executes this...
The stale-state pattern

A controller should correctly handle staleness caused by asynchrony and caching.

1. Inject delay to make a backup cache stale
2. Reconnect the controller to the stale cache

Reference run

S1
S2
S3
S4
S5

Perturbed run

S1
S2
S3
S1 (replayed in the controller’s view)
The unobserved-state pattern

A controller should function correctly without observing every state.

Inject delay to make the controller miss a state

Reference run

Perturbed run

S1
S2
S3
S4
S5

S1
S2
S4
S5

S3 missed in the controller’s view
Exhaustive perturbation for each pattern

• **Key principle:** Inject faults at each execution point

• Run many different tests, each performing a different perturbation
  • Intermediate-state: Crash after *every* state update
  • Stale-state: Replay *every* stale state
  • Unobserved-state: Make the controller miss *every* state
Prune ineffective perturbations for efficiency

**Key principle:** Prune out perturbations that cannot affect a controller’s behavior

- **Intermediate-state:** Prune out crashes that do not result in new intermediate states
- **Stale- and unobserved-state:** Avoid perturbing the state if observing the state does not causally lead to any controller effect
  - Reason about **causality** from state to effect

```java
// Reconcile cycle
... Create(...)...
... Update(...)...
Delete(...)...
... Update(...)...
```

Delete a non-existing object
Sieve end-to-end workflow

Input

Controller

Build & deploy scripts

Test workloads

1. Produce a reference run

2. Generate test plans

A test plan describes a concrete perturbation

3. Produce a perturbed run for each test plan

4. Flag bugs with differential oracles

Output

Test results for each perturbation
Evaluation

• Applied Sieve to 10 popular Kubernetes controllers

• Can Sieve **effectively** find new bugs in real-world controllers?
  • Sieve found 46 bugs in 10 controllers

• Does Sieve do so **efficiently**?
  • Sieve pruned out 46% - 99% of perturbations
  • Sieve tested each controller with a **nightly** run

• Are Sieve’s testing results **trustworthy**?
  • Sieve had a low false positive rate of 3.5%
## Finding new bugs

<table>
<thead>
<tr>
<th>Controller</th>
<th>Intermediate state bugs</th>
<th>Stale state bugs</th>
<th>Unobserved state bugs</th>
<th>Indirect bugs</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>cass-operator</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>cassandra-operator</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>casskop</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>elastic-operator</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>mongodb-operator</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>nifikop</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>rabbitmq-operator</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>xtradb-operator</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>yogabyte-operator</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>zookeeper-operator</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>11</strong></td>
<td><strong>19</strong></td>
<td><strong>7</strong></td>
<td><strong>9</strong></td>
<td><strong>46</strong></td>
</tr>
</tbody>
</table>
Conclusion

• Controller reliability is critical but challenging!

• Sieve: automatic reliability testing for Kubernetes controllers
  • **Key idea:** Perturbing the controller’s view of the cluster state
  • **Usability:** Testing unmodified controllers
  • **Reproducibility:** Reproducing detected bugs reliably

• **Open sourced** at [https://github.com/sieve-project/sieve](https://github.com/sieve-project/sieve)
  • Test your controller with Sieve!