CrashTuner: Detecting Crash Recovery Bugs in Cloud Systems via Meta-info Analysis

Jie Lu, Chen Liu, Lian Li, Xiaobing Feng, Feng Tan, Jun Yang, Liang You

Institute of Computing Technology, Chinese Academy of Sciences

University of Chinese Academy of Sciences. Alibaba Group
Crash Recovery

- Recovery must be a first-class operation of distributed systems¹.

---

Recovery must be a first-class operation of distributed systems\(^1\).

- Nodes can crash due to different reasons.\(^2\)


Recovery must be a first-class operation of distributed systems\(^3\).

Node Crash Events can be common in a large cluster (At least 180).\(^4\)

---


Crash Recovery Code can be buggy and often result in catastrophic failure.\textsuperscript{5}

---


Crash Reversal Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.\(^5\)
- Existing detection approaches

---


Crash Recovery Code can be buggy and often result in catastrophic failure.\(^5\)

- Existing detection approaches
  - Random fault injection: Ineffective\(^6\).

---


Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.\(^5\)
- Existing detection approaches
  - Random fault injection: Ineffective\(^6\).
  - Model checking: Inefficient and requires manual specifications\(^7\).

---


Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.\(^5\)
- Existing detection approaches
  - Random fault injection: Ineffective\(^6\).
  - Model checking: Inefficient and requires manual specifications\(^7\).
- Crash-Recovery bugs still widely exist in distributed system.\(^8\)

---


Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.\(^5\)
- Existing detection approaches
  - Random fault injection: Ineffective\(^6\).
  - Model checking: Inefficient and requires manual specifications\(^7\).
- Crash-Recovery bugs still widely exist in distributed system.\(^8\)
  - Distributed systems have large state space to explore.

---


Crash-Recovery Bugs and Detection

- Crash Recovery Code can be buggy and often result in catastrophic failure.\(^5\)

- Existing detection approaches
  - Random fault injection: Ineffective\(^6\).
  - Model checking: Inefficient and requires manual specifications\(^7\).

- Crash-Recovery bugs still widely exist in distributed system.\(^8\)
  - Distributed systems have large state space to explore.
  - Crash-Recovery bugs can only be triggered when nodes crash under special timing conditions.


A new approach to automatically detect crash-recovery bugs in distributed systems.

- **21** new crash-recovery bugs (including **10** critical bugs).
- Test **5** distributed systems in **35** hours.
How does CrashTuner do it?
## Findings

- Existing Crash-Recovery bugs can be easily triggered when nodes:

  - Crash before reading variables
  - Crash after writing variables

  One thing in common: All these variables are meta-info variables.

---

**Figure:** 116 Crash-Recovery Bugs from four distributed Systems.

<table>
<thead>
<tr>
<th>Distributed File System</th>
<th>Distributed Resource Manager System</th>
<th>Distributed Database</th>
<th>Centralized service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master/slave</td>
<td>Master/slave</td>
<td>Master/slave</td>
<td>Master/slave based on leader election</td>
</tr>
</tbody>
</table>
Findings

- Existing Crash-Recovery bugs can be easily triggered when nodes:
  - Crash before reading variables

---

Figure: 116 Crash-Recovery Bugs from four distributed Systems.

<table>
<thead>
<tr>
<th>System</th>
<th>Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td>Master/slave</td>
</tr>
<tr>
<td>YARN</td>
<td>Master/slave</td>
</tr>
<tr>
<td>HBASE</td>
<td>Master/slave</td>
</tr>
<tr>
<td>Zookeeper</td>
<td>Centralized service</td>
</tr>
<tr>
<td>Master/slave based on</td>
<td>leader election</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Findings

- Existing Crash-Recovery bugs can be easily triggered when nodes:
  - Crash before reading variables
  - Crash after writing variables.

**Figure:** 116 Crash-Recovery Bugs from four distributed Systems.

<table>
<thead>
<tr>
<th>Distributed File System</th>
<th>Distributed Resource Manager System</th>
<th>Distributed Database</th>
<th>Centralized service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master/slave</td>
<td>Master/slave</td>
<td>Master/slave</td>
<td>Master/slave based on leader election</td>
</tr>
</tbody>
</table>
Findings

- Existing Crash-Recovery bugs can be easily triggered when nodes:
  - Crash before reading variables
  - Crash after writing variables.

- One thing in common: All these variables are **meta-info** variables.

**Figure:** 116 Crash-Recovery Bugs from four distributed Systems.

<table>
<thead>
<tr>
<th>Distributed File System</th>
<th>Distributed Resource Manager System</th>
<th>Distributed Database</th>
<th>Centralized service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master/slave</td>
<td>Master/slave</td>
<td>Master/slave</td>
<td>Master/slave based on leader election</td>
</tr>
</tbody>
</table>
What are meta-info variables?

A simplified YARN example
Job_1
  └── Application_1
      └── AppAttempt_1
Meta-info: Nodes, Jobs, Tasks, Applications, Containers, Attempt, Session…

Abstrated state
Instance of abstraction
Meta-info value: Node_1, Node_m, Job_1, task_m……
High Level System State
Node Crashes before Reading meta-info variables
New Bug (YARN-9238) detected by CrashTuner

YARN@Node1

Recovery

Task1@Node2
New Bug (YARN-9238) detected by CrashTuner

YARN@Node1

meta-info variable
task_1

Recovery

Task1@Node2
New Bug (YARN-9238) detected by CrashTuner

- **YARN@Node1**
  - meta-info variable task_1

- **Task1@Node2**

(1)detection
New Bug (YARN-9238) detected by CrashTuner
New Bug (YARN-9238) detected by CrashTuner

YARN@Node1

- meta-info variable task_1
- uninitialized task_2

Recovery

Task1@Node2

(1) detection

(2) update

(3) read
New Bug (YARN-9238) detected by CrashTuner

YARN@Node1

- meta-info variable task_1
- uninitialized task_2

Recovery

Task1@Node2

allocate

(1) detection

(2) update

(3) read
How CrashTuner Detected it?

Inject sleep and crash before reading the variable

YARN@Node1

Recovery

Task1@Node2

(1)read

task_1
How CrashTuner Detected it?

Inject sleep and crash before reading the variable

YARN@Node1

Recovery

Task1@Node2

Sleep

(1)read

task_1
How CrashTuner Detected it?

Inject sleep and crash before reading the variable

YARN@Node1

Sleep

uninitialized

(1)read

(2)update

uninitialized

(1)read

Task1@Node2

Recovery
How CrashTuner Detected it?

Inject sleep and crash before reading the variable

YARN@Node1

Recovery

Task1@Node2

uninitialized task_2

(1)read

(2)update
Node Crashes after writing meta-info variables
New Bug (HBASE-22041) detected by CrashTuner

HMaster@node1

meta-info variable
onlineservers

CluterTracker

Recovery

Zookeeper

slave@node2
New Bug (HBASE-22041) detected by CrashTuner
New Bug (HBASE-22041) detected by CrashTuner

HMaster@node1

CluterTracker

meta-info variable
onlineServers

(1) heartbeat

(2) write

Recovery

Zookeeper

slave@node2

(3) register

(4) crash

(5) remove

(6) read
New Bug (HBASE-22041) detected by CrashTuner

HMaster@node1

ClusterTracker

slave@node2

meta-info variable
onlineServers

(1) heartbeat

(2) write

(3) register

Recovery

Zookeeper

(4) crash

(5) remove

(6) read
New Bug (HBASE-22041) detected by CrashTuner

- HMaster@node1
- CluterTracker
- slave@node2

- meta-info variable onlineServers
- (1) heartbeat
- (2) write
- (3) register
- (4) crash
- (5) remove
- Recovery
- Zookeeper

Jie Lu, Chen Liu, Lian Li, Xiaobing Feng Feng Tan, Jun Yang, Liang You | ICT
New Bug (HBASE-22041) detected by CrashTuner

HMaster@node1

CluterTracker

slave@node2

meta-info variable
onlineServers

(1)heartbeat

(2)write

(5)remove

(4)crash

(3)register

Recovery

Zookeeper
New Bug (HBASE-22041) detected by CrashTuner

```
while (onlineSevers.contains(rs)) {
    try {
        connect(rs);
        catch (Exception e) {
            log.info("Retry");
        }
    }
}
```
How CrashTuner detected it

Inject crash after writing the variable

HMaster@node1

| onlineServers |

CluterTracker

slave@node2

(1) heartbeat
(1) write
(6) read
Inject crash after writing the variable

HMaster@node1  CluterTracker  slave@node2

onlineServers

(1)write  (1)heartbeat
How CrashTuner detected it

Inject crash after writing the variable

HMaster@node1 -> onlineServers

CluterTracker

(1)write

(1)heartbeat

slave@node2
How CrashTuner detected it

Inject crash after writing the variable

HMast@node1

onlineServers

(6) read

CluterTracker

(1) write

(1) heartbeat

slave@node2
Meta-info variable identification

How to find meta-info variables?
Meta-info variable Identification

Node referencing variables are meta-info variables.

LOG.info("NodeManager from node " + address + " is assigned " + nodeId)
Node referencing variables are meta-info variables.

\[
\text{LOG.info("NodeManager from node " + address + " is assigned " + nodeId)}
\]

NodeManager from node (.*) is assigned (.*)
META-INFO variable Identification

Node referencing variables are meta-info variables.

LOG.info("NodeManager from node "+ address + " is assigned "+ nodeId)
Meta-info variable Identification

Node referencing variables are meta-info variables.

```java
LOG.info("NodeManager from node \(.*\) is assigned \(.*\)")
```

Hostname, meta-info value

hadoop14 is assigned hadoop14:8088
Meta-info variable Identification

Node referencing variables are meta-info variables.

LOG.info("NodeManager from node \(.*\) + address + \(.*\) is assigned \(.*\) + nodeId)
Variables related to meta-info variable are meta-info variables. Appearing in a same log instance.

```java
LOG.info("Assigned Container" + containerId + " on host " + nodeId)
```

```java
LOG.info("Assigned Container" + containerId + " to " + attempt)
```
Variables related to meta-info variable are meta-info variables. Appearing in a same log instance.

```
LOG.info("Assigned Container" + containerId + " on host " + nodeId)
```

```
LOG.info("Assigned Container" + containerId + " to " + attempt)
```

```
LOG.info("Assigned Container" + containerId + " on host " + nodeId)
```
Variables related to meta-info variable are meta-info variables. Appearing in a same log instance.

```
LOG.info("Assigned Container" + containerId + " on host " + nodeId)
```

```
LOG.info("Assigned Container" + containerId + " to " + attempt)
```
Variables related to meta-info variable are meta-info variables. Appearing in a same log instance.

```
LOG.info("Assigned Container" + containerId + " on host " + nodeId)
```

```
LOG.info("Assigned Container" + containerId + " to " + attempt)
```
Type based static analysis to discover meta-info variables not logged.

```java
/* - tracks the state of all cluster nodes */
public class ClusterNodeTracker<N extends SchedulerNode> {
    private HashMap<NodeId, N> nodes = new HashMap<>();
}
```

**Meta-info type**  
**Meta-info variable**
- Pre-read points of meta-info variables.
Pre-read points of meta-info variables.

Post-write points of meta-info variables.
CrashTuner

**Node to Crash**

---

**Crash Points**

- Crash Point Identification
  - Log analysis
  - Static Crash Point Analysis
  - Profile

---

**Fault Injection Testing**

- Online Log Analysis
  - Runtime Logs
  - Runtime meta-infos
- Meta-info Manager
  - meta-info value
  - Node to crash
- Trigger
  - bugs
Which node to Crash?

Job@Node1  

Crash Point

read or write

Task1@Node2

Target Node

Crash node2 at the crash point in node1.
Inferring the Target Node

Run time logs
Assigned Container_1 on hadoop14:80
Assigned Container_1 to attempt_1
Assigned Container_2 on hadoop15:80
Assigned Container_2 to attempt_2

Container_1 and attempt_1 on hadoop14
Container_2 and attempt_2 on hadoop15
Inferring the Target Node

Run time logs
Assigned Container_1 on hadoop14:80
Assigned Container_1 to atemmppt_1
Assigned Container_2 on hadoop15:80
Assigned Container_2 to atemmppt_2

Online log analysis
Logstash
Regular Expression Filter
  Container_(.*)
  hadoop(.*):(.*)
  Attempt_(.*)
Inferring the Target Node

Run time logs
- Assigned Container_1 on hadoop14:80
- Assigned Container_1 to atemmppt_1
- Assigned Container_2 on hadoop15:80
- Assigned Container_2 to atemmppt_2

Online log analysis
Logstash
Regular Expression Filter
- Container_(.*)
- hadoop\((.*)\):(\.*)
- Attempt_(.*)

Meta-info Manager

<table>
<thead>
<tr>
<th>Key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>container_1</td>
<td>hadoop14:80</td>
</tr>
<tr>
<td>attempt_1</td>
<td>hadoop14:80</td>
</tr>
<tr>
<td>container_2</td>
<td>hadoop15:80</td>
</tr>
<tr>
<td>attempt_2</td>
<td>hadoop15:80</td>
</tr>
</tbody>
</table>
Inferring the Target Node

Run time logs
Assigned Container_1 on hadoop14:80
Assigned Container_1 to attempt_1
Assigned Container_2 on hadoop15:80
Assigned Container_2 to attempt_2

Online log analysis
Logstash
Regular Expression Filter
Container_(.*), hadoop.(.*):(.*), Attempt_(.*)

Trigger
1. shutdown(<context>, id)
2. wait()
3. value=nodes.get(id)

Meta-info Manager
<table>
<thead>
<tr>
<th>Key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>container_1</td>
<td>hadoop14:80</td>
</tr>
<tr>
<td>attempt_1</td>
<td>hadoop14:80</td>
</tr>
<tr>
<td>container_2</td>
<td>hadoop15:80</td>
</tr>
<tr>
<td>attempt_2</td>
<td>hadoop15:80</td>
</tr>
</tbody>
</table>
### Evaluations

**Table:** Five distributed Systems under testing (Cassandra is not our bug-studied system).

<table>
<thead>
<tr>
<th>System</th>
<th>Configure Change</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop2/Yarn</td>
<td>enable opportunistic</td>
<td>Wordcount</td>
</tr>
<tr>
<td>HDFS</td>
<td>—</td>
<td>TestDFSIO, curl</td>
</tr>
<tr>
<td>HBase</td>
<td>—</td>
<td>PE, curl</td>
</tr>
<tr>
<td>Zookeeper</td>
<td>—</td>
<td>Smoketest</td>
</tr>
<tr>
<td>Cassandra</td>
<td>—</td>
<td>Stress</td>
</tr>
</tbody>
</table>
## Evaluations

*Table:* The number of meta-info and crash point and test time.

<table>
<thead>
<tr>
<th>System</th>
<th># Meta-info</th>
<th># Crash Points</th>
<th>Test time(h)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Types</td>
<td>Fields</td>
<td>Access Points</td>
</tr>
<tr>
<td>Hadoop2/Yarn</td>
<td>107</td>
<td>1,251</td>
<td>5,109</td>
</tr>
<tr>
<td>HBase</td>
<td>34</td>
<td>733</td>
<td>4,032</td>
</tr>
<tr>
<td>HDFS</td>
<td>43</td>
<td>315</td>
<td>1,924</td>
</tr>
<tr>
<td>ZooKeeper</td>
<td>3</td>
<td>13</td>
<td>90</td>
</tr>
<tr>
<td>Cassandra</td>
<td>1</td>
<td>122</td>
<td>666</td>
</tr>
<tr>
<td>total</td>
<td>188</td>
<td>2,434</td>
<td>11,821</td>
</tr>
</tbody>
</table>
CrashTuner reduces 99.91% unnecessary crash points

**Table:** The number of meta-info and crash point and test time.

<table>
<thead>
<tr>
<th>System</th>
<th># Meta-info Types</th>
<th># Meta-info Fields</th>
<th># Crash Points Access Points</th>
<th># Crash Points Static</th>
<th># Crash Points Dynamic</th>
<th>Test time(h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop2/Yarn</td>
<td>107</td>
<td>1,251</td>
<td>5,109</td>
<td>1,524</td>
<td>453</td>
<td>17.39</td>
</tr>
<tr>
<td>HBase</td>
<td>34</td>
<td>733</td>
<td>4,032</td>
<td>920</td>
<td>257</td>
<td>8.27</td>
</tr>
<tr>
<td>HDFS</td>
<td>43</td>
<td>315</td>
<td>1,924</td>
<td>495</td>
<td>237</td>
<td>8.65</td>
</tr>
<tr>
<td>ZooKeeper</td>
<td>3</td>
<td>13</td>
<td>90</td>
<td>41</td>
<td>40</td>
<td>0.27</td>
</tr>
<tr>
<td>Cassandra</td>
<td>1</td>
<td>122</td>
<td>666</td>
<td>197</td>
<td>69</td>
<td>1.10</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>188</strong></td>
<td><strong>2,434</strong></td>
<td><strong>11,821</strong></td>
<td><strong>3,177</strong></td>
<td><strong>1,056</strong></td>
<td><strong>35.68</strong></td>
</tr>
</tbody>
</table>
CrashTuner reports 21 new bugs, 16 of them are already fixed

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>Type</th>
<th>Status</th>
<th>Symptom</th>
<th>Meta-info</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN-1</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Invalid event for current state of ApplicationAttempt</td>
<td>ContainerId</td>
</tr>
<tr>
<td>YARN-2</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Invalid event for current state of ApplicationAttempt</td>
<td>ApplicationId</td>
</tr>
<tr>
<td>YARN-3</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Invalid event for current state of ApplicationAttempt</td>
<td>ContainerId</td>
</tr>
<tr>
<td>YARN-4</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Invalid event for current state of ApplicationAttempt</td>
<td>ContainerId</td>
</tr>
<tr>
<td>YARN-5(2)</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Invalid event for current state of ApplicationAttempt</td>
<td>ApplicationAttemptId</td>
</tr>
<tr>
<td>YARN-7(2)</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Invalid event for current state of Container</td>
<td>ApplicationId</td>
</tr>
<tr>
<td>YARN-9</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Invalid event for current state of Container</td>
<td>NodeId</td>
</tr>
<tr>
<td>YARN-10</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Invalid event for current state of Container</td>
<td>NodeId</td>
</tr>
<tr>
<td>YARN-11</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Invalid event for current state of Container</td>
<td>ContainerId</td>
</tr>
<tr>
<td>HBASE-12</td>
<td>post-write</td>
<td>Fixed</td>
<td>Invalid event for current state of Container</td>
<td>ApplicationId</td>
</tr>
<tr>
<td>HBASE-13</td>
<td>pre-read</td>
<td>Unresolved</td>
<td>Atomic violation causing shutdown fails</td>
<td>ApplicationId</td>
</tr>
<tr>
<td>HBASE-14</td>
<td>post-write</td>
<td>Unresolved</td>
<td>Master startup hang and print thousands of logs</td>
<td>RegionInfo</td>
</tr>
<tr>
<td>HBASE-15</td>
<td>post-write</td>
<td>Unresolved</td>
<td>Shutdown before initialization causing abort</td>
<td>ServerName</td>
</tr>
<tr>
<td>HBASE-16</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Master Fails to become active due to LeaseException</td>
<td>ServerName</td>
</tr>
<tr>
<td>HDFS-17</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Shutdown before register causing abort</td>
<td>ServerName</td>
</tr>
<tr>
<td>HDFS-18(2)</td>
<td>pre-read</td>
<td>Fixed</td>
<td>Request fails due to removed node</td>
<td>DataNodeInfo</td>
</tr>
<tr>
<td>MR-20</td>
<td>post-write</td>
<td>Unresolved</td>
<td>Shutdown before initialization causing abort</td>
<td>TaskAttemptId</td>
</tr>
<tr>
<td>CA-21</td>
<td>pre-read</td>
<td>Unresolved</td>
<td>Request fails due to removed node</td>
<td>InetAddressAndPort</td>
</tr>
</tbody>
</table>
Comparing to other fault injection strategies

CrashTuner report one bug in 50.29 runs within 1.70 hours.

- Random fault injection: 3 bugs, 1 bug per 5000 runs within 90.83 hours
- IO around crash injection, 1 bugs, 1 bug per 4500 runs within 156.88 hours
- All bugs can be detected by CrashTuner.
Comparing to other fault injection strategies

CrashTuner report one bug in 50.29 runs within 1.70 hours.

- Random fault injection: 3 bugs, 1 bug per 5000 runs within 90.83 hours
- IO around crash injection, 1 bugs, 1 bug per 4500 runs within 156.88 hours
- All bugs can be detected by CrashTuner.

CrashTuner is much more Efficient and Effective than random crash injection and IO around crash injection
Limitations and Future Work

- CrashTuner maybe not good enough to test system with Bad Log Quality.
  - Developer can annotate the meta-info type.

- CrashTuner only inject one crash.
  - We can extend CrashTuner to test two or more crash events.

- CrashTuner only test Java based system.
  - Our study on k8s (implemented with Golang) shows that it also have meta-info related crash-recovery bugs.
  - We are extending CrashTuner to work with System written by Golang and C++.
Relate Works

- Crash-recovery bug studies.
  - CBSDB\(^9\), TaxDC\(^{10}\), CREB\(^{11}\)

- Crash-recovery bug detection
  - Fault injection: Fate\(^{12}\), Fcatch\(^{13}\)

- Log analysis for distribute systems
  - Stitch[OSDI2016], lprof[OSDI2014]

---


Abstraction is so fundamental that sometimes we forget its importance!\textsuperscript{14}

—Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

Conclusion

Abstraction is so fundamental that sometimes we forget its importance!¹⁴
—Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

Meta-info is a well-suited abstraction for distributed systems!

Thank you! Any Questions?
backup slides
How to find the meta-info value at crash point

```java
private void launch() throws IOException, YarnException {
    ContainerId masterContainerID = masterContainer.getId();
    ApplicationSubmissionContext applicationContext = application.getSubmissionContext();
    LOG.info("Setting up container " + masterContainer + " for AM " + application.getAppAttemptId());
}

public ContainerId getId() {
    return this.containerId;
}
```
public ContainerId getId() {
    ContainerProtoOrBuilder p = viaProto ? proto : builder;
    if (this.containerId != null) {
        return this.containerId;
    }
    if (!p.hasId()) {
        return null;
    }
    this.containerId = convertFromProtoFormat(p.getId());
    return this.containerId;
}
How to find the meta-info value at crash point

```java
private void launch() throws IOException, YarnException {
    ContainerId masterContainerID = masterContainer.getId();

    ApplicationSubmissionContext applicationContext = application.getSubmissionContext();

    LOG.info("Setting up container " + masterContainer + " for AM " + application.getAppAttemptId());
}
```
public void setNodeId(NodeId nodeId) {
    lock();
    this.nodeId = nodeId;
    unlock();
}

public NodeId getNodeId() {
    lock();
    return nodeId;
    unlock();
}

public void lunch()

    //Crash Point
   NodeId nodeId = getNodeId();
    node.getHttpAddress();
For Event Handler
CrashTuner can be easily embedded in CI/CD Pipeline at Auto Test phase.
Why Random fault injection is less Efficient and Effective?

Random approach is less time sensitive.
New bug: YARN-9238
Discussion

Why Random fault injection is less Efficient and Effective?

Random approach is less time sensitive.
New bug: YARN-9238

Recovery  
JobAttempt

fail

READ

Jie Lu, Chen Liu, Lian Li, Xiaobing Feng, Feng Tan, Jun Yang, Liang You | ICT
Discussion

Why Random fault injection is less Efficient and Effective?

Random approach is less time sensitive.
New bug: YARN-9238

Recovery  JobAttempt

sanity check  fail

READ  fail
Discussion

Why Random fault injection is less Efficient and Effective?

Random approach is less time sensitive.
New bug: YARN-9238
Discussion

Why Random fault injection is less Efficient and Effective?

Random approach is less time sensitive.
New bug: YARN-9238
Discussion

Why IO around Fault injection is less Efficient and Effective?
Discussion

Why IO around Fault injection is less Efficient and Effective?

- IO Crash Points are far away the real crash point.
- Local recovery: Developers instinctively handle the error related IO operation.

```java
1  try {
2       connect();
3  } catch (Exception e){
4       //carefully handle
5  }
```
Bug example of shutdown: HBASE-22017

Shutdown can simulate software failure [Socc2014] and gray failure [OSDI2018, ATC2019]: one component is sick, left are health.

HMaster

RPCServer

LeaseManager

RegionServer

(1) shutdown
(2) scan
(3) request
(4) fail
(5) LeaseException
Bug example of shutdown: HBASE-22017

Shutdown can simulate software failure [Socc2014] and gray failure [OSDI2018, ATC2019]: one component is sick, left are health.

1. shutdown
2. scan
3. request
4. fail
5. LeaseException
Bug example of shutdown: HBASE-22017

Shutdown can simulate software failure [Socc2014] and gray failure [OSDI2018, ATC2019]: one component is sick, left are health.

HMaster

RPCServer

LeaseManager

RegionServer

(1) shutdown

(2) scan
Bug example of shutdown: HBASE-22017

Shutdown can simulate software failure [Socc2014] and gray failure [OSDI2018, ATC2019]: one component is sick, left are health.

(1) shutdown
(2) scan
(3) request
(4) fail

HMaster

RPCServer

RegionServer

LeaseManager
Bug example of shutdown:HBASE-22017

Shutdown can simulate software failure[Socc2014] and gray failure[OSDI2018, ATC2019]: one component is sick, left are health.
Bug example of shutdown:HBASE-22017

Shutdown can simulate software failure[SoCC 2014] and gray failure[OSDI 2018, ATC 2019]: one component is sick, left are health.

\[
\text{HMaster} \rightarrow \text{RPCServer} \rightarrow \text{RegionServer} \rightarrow \text{LeaseManager}
\]

1. shutdown
2. scan
3. request
4. fail
5. LeaseException
Q1: Profile: Dynamic Crash Point

- Filtering un-executed Static Crash Points.

- Calling context of One Crash Point.

Crash Point = \{CLASS, Method, LineNumber, PreOrPost, Call Stack\}