DETECTING FAILURES IN DISTRIBUTED SYSTEMS WITH THE FALCON SPY NETWORK

JOSHUA LENERS, HAO WU, WEI-LUN HUNG, MARCOS AGUILERA, MICHAEL WALFISH

EECS 591 PRESENTER  YUWEI BAO
404
Page not found
We are sorry but the page you are looking for does not exist.

• Application crashes
• Operating system crashes
• Device driver crashes
• Application deadlocks
• Application livelocks
• Hardware failures

FAILURE!
THE IDEAL FAILURE DETECTOR

• FAST DETECTION
• RELIABLE
• LITTLE DISRUPTION
• INEXPENSIVE

A reliable failure detector (RFD):

To report process \( p \) as UP or DOWN

• if the RFD reports \( p \) as DOWN, then \( p \) has crashed;

• if \( p \) crashes, then the RFD eventually reports \( p \) as DOWN (and does so ever after).
INSPIRATION

• Process core dumps
• OS panics
• Machine loses power

Application
OS
VMM
Network Switch

• Disappear from the process table
• Stop scheduling processes
• Stop communicating with network switch

Chained spy network + local timeouts
ARCHITECTURE OF FALCON

Figure 1

4 layers of spies

Application
OS
VMM
Network Switch

Client RFD interface
DESIGN PRINCIPLES

• Make it reliable
• Avoid end-to-end timeouts
• Peek inside the layers
• Kill surgically, if needed
• Monitor the monitors

Target settings: large data centers, enterprise systems

Assumptions:
• Permission to (limited) modifications to the software stack
• Users are trustworthy
• Access control are orthogonal

Limitations:
• Blocked under network partition
• Single data center
• No Byzantine failures
## FALCON – CLIENT RFD INTERFACE

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>init(target)</code></td>
<td>register with spies</td>
</tr>
<tr>
<td><code>uninit()</code></td>
<td>deregister with spies</td>
</tr>
<tr>
<td><code>query()</code></td>
<td>query the operational status</td>
</tr>
<tr>
<td><code>set_callback(callback)</code></td>
<td>install callback function</td>
</tr>
<tr>
<td><code>clear_callback()</code></td>
<td>cancel callback function</td>
</tr>
<tr>
<td><code>start_timeout(timeout)</code></td>
<td>start end-to-end timeout timer</td>
</tr>
<tr>
<td><code>stop_timeout()</code></td>
<td>stop end-to-end timeout timer</td>
</tr>
</tbody>
</table>

### Figure 1

- **App1**
- **App2**
- **OS**
- **VMM**
- **client library**

### Figure 2

- Spy
- Call or response
- Monitoring

- "Up" arrow
- "Down" arrow
FALCON - SPIES

- Register()
- Cancel()
- Kill()
- Callback()
- Monitor()

- End-to-end timeout & kills

Figure 6

```
while true
    sense layer and set rc accordingly
    if rc = CERTAINLY_DOWN then
        callback(LAYER_DOWN)
    if rc = CERTAINLY_UP then
        if have not called callback within UP-INTERVAL then
            callback(LAYER_UP)
    if rc = SUSPECT_CRASH then
        kill()
        callback(LAYER_DOWN)
```

Figure 3
## FALCON - SPIES

### Components

<table>
<thead>
<tr>
<th>Application</th>
<th>OS</th>
<th>VMM</th>
<th>Network Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application specific ( f() )</td>
<td>Incrementer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Detection

| | \( T_{\text{app-check}} = 100\text{ms} \) | \( T_{\text{OS-inc}} = 1\text{ms} \) | \( T_{VMM\text{-check}} \) |
| | \( T_{\text{app-resp App Specf}} \) | \( T_{\text{OS-check}} = 100\text{ms} \) | \( T_{VMM\text{-check-2}} = 5\text{ms} \) |
| | \( T_{\text{OS-resp}} = 100\text{ms} \) | | \( T_{VMM\text{-resp}} = 20\text{ms} \) |
| | | \( N_{VMM\text{-retry}} = 5 \) | |
| | | Total waiting time | |
| | | \( T_{VMM\text{-resp}}(N_{VMM\text{-retry}}+1) = 120\text{ms} \) | |

### Reliability

| | Process table | Scheduling process? | Communication to switch? | End-to-End timeout & kill |
| | | | | |

Reliability process table

Communication to switch?

End-to-End timeout & kill
APPLICATION SPIES

• Inspector: thread inside the application f()
• Enforcer: distinguished high-priority process in OS
• Communication: inter-process communication (IPC) channel
• -f() reports down
  - Tapp-check = 100ms, Tapp-resp (= 100ms CPU time)
• Suspects a crash: process table / kill & conf
OS SPIES

• Inspector: kernel module, an incrementer
• Enforcer: module inside the VMM
• Communication: infer by counter
• \textbf{TOS-check} = 100ms, \textbf{TOS-resp} = 100ms
• Suspects a crash: VMM to stop scheduling & conf
VMM SPIES

• Inspector: module in the VMM.
• Enforcer: software module on the switch
• Communication: infer by switch
• $TVMM\text{-check, TVMM\text{-check-2}} = 5s$, $TVMM\text{-resp} = 20ms$, $NVMM\text{-retry} = 5$
• $TVMM\text{-resp} \cdot (NVMM\text{-retry} + 1) = 120$ ms
• Suspects a crash: shut down the network port
### EVALUATION

#### Applications
- ZooKeeper
- ZooKeeper + Falcon
- minimal Paxos-based replication library
- minimal Paxos-based replication library + Falcon
- primary-backup-based replication library + Falcon

<table>
<thead>
<tr>
<th>Failure</th>
<th>Action Taken by Falcon</th>
</tr>
</thead>
<tbody>
<tr>
<td>app crash</td>
<td>app enforcer detects failure</td>
</tr>
<tr>
<td>app layer-down report</td>
<td>app enforcer kills application</td>
</tr>
<tr>
<td>app inspector hangs</td>
<td>app enforcer kills application</td>
</tr>
<tr>
<td>kernel hang</td>
<td>OS enforcer kills guest OS</td>
</tr>
<tr>
<td>kernel stack overflow</td>
<td>OS enforcer kills guest OS</td>
</tr>
<tr>
<td>kernel panic</td>
<td>OS enforcer kills guest OS</td>
</tr>
<tr>
<td>VMM error / guest exit</td>
<td>OS enforcer detects failure</td>
</tr>
<tr>
<td>host down</td>
<td>VMM enforcer kills VMM/host</td>
</tr>
<tr>
<td>crashed app enforcer + app crash</td>
<td>E2E timeout kills guest OS</td>
</tr>
<tr>
<td>crashed incrementer</td>
<td>OS enforcer kills guest OS</td>
</tr>
<tr>
<td>crashed OS enforcer + OS crash</td>
<td>E2E timeout kills VMM/host</td>
</tr>
<tr>
<td>crashed VMM inspector</td>
<td>VMM enforcer kills VMM/host</td>
</tr>
</tbody>
</table>

#### Transient Condition

<table>
<thead>
<tr>
<th>Transient Condition</th>
<th>Action Taken by Falcon</th>
</tr>
</thead>
<tbody>
<tr>
<td>hung system call</td>
<td>none</td>
</tr>
<tr>
<td>CPU contention within guest</td>
<td>none</td>
</tr>
<tr>
<td>CPU contention across guests</td>
<td>none</td>
</tr>
<tr>
<td>memory contention within guest</td>
<td>none</td>
</tr>
<tr>
<td>memory contention across guests</td>
<td>OS enforcer kills guest OS</td>
</tr>
<tr>
<td>packet flood between guests</td>
<td>none</td>
</tr>
<tr>
<td>packet flood between VMMs</td>
<td>VMM enforcer kills VMM/host</td>
</tr>
</tbody>
</table>
EVALUATION – FAST DETECTION?

Figure 11

- Falcon is fast
- The processing delay longer than expected
- Still, fast, an order of magnitude faster
- No network delay
EVALUATION -- AVAILABILITY

Availability: Median response gap of ZooKeeper

- 4 nodes
- 3 server + 1 client
- 1 leader + 2 followers

- ZK reacts quickly on app crash
- Otherwise Falcon is faster
- PMP similar results

Figure 12
**EVALUATION -- DISRUPTION**

- Reactions to failures match expectations
- Reactions to transient conditions match most of the time except for 2 cases
- Seriously we really don’t want to kill

<table>
<thead>
<tr>
<th>failure</th>
<th>action taken by Falcon</th>
</tr>
</thead>
<tbody>
<tr>
<td>app crash</td>
<td>app enforcer detects failure</td>
</tr>
<tr>
<td>app layer-down report</td>
<td>app enforcer kills application</td>
</tr>
<tr>
<td>app inspector hangs</td>
<td>app enforcer kills application</td>
</tr>
<tr>
<td>kernel hang</td>
<td>OS enforcer kills guest OS</td>
</tr>
<tr>
<td>kernel stack overflow</td>
<td>OS enforcer kills guest OS</td>
</tr>
<tr>
<td>kernel panic</td>
<td>OS enforcer kills guest OS</td>
</tr>
<tr>
<td>VMM error / guest exit host down</td>
<td>OS enforcer detects failure</td>
</tr>
<tr>
<td></td>
<td>VMM enforcer kills VMM/host</td>
</tr>
<tr>
<td>crashed app enforcer + app crash</td>
<td>E2E timeout kills guest OS</td>
</tr>
<tr>
<td>crashed incrementer</td>
<td>OS enforcer kills guest OS</td>
</tr>
<tr>
<td>crashed OS enforcer + OS crash</td>
<td>E2E timeout kills VMM/host</td>
</tr>
<tr>
<td>crashed VMM inspector</td>
<td>VMM enforcer kills VMM/host</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>transient condition</th>
<th>action taken by Falcon</th>
</tr>
</thead>
<tbody>
<tr>
<td>hung system call</td>
<td>none</td>
</tr>
<tr>
<td>CPU contention within guest</td>
<td>none</td>
</tr>
<tr>
<td>CPU contention across guests</td>
<td>none</td>
</tr>
<tr>
<td>memory contention within guest</td>
<td>none</td>
</tr>
<tr>
<td>memory contention across guests</td>
<td>OS enforcer kills guest OS</td>
</tr>
<tr>
<td>packet flood between guests</td>
<td>none</td>
</tr>
<tr>
<td>packet flood between VMMs</td>
<td>VMM enforcer kills VMM/host</td>
</tr>
</tbody>
</table>

Figure 13
EVALUATION -- COST

- CPU overhead is basically very low
- Limited lines of extra code
- Simplify replication approach: Paxos → Primary-backup

### Figure 14

<table>
<thead>
<tr>
<th>component ($4$)</th>
<th>CPU overhead (percent of a core’s cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>app uses no CPU</td>
</tr>
<tr>
<td>app inspector</td>
<td>0.06</td>
</tr>
<tr>
<td>app enforcer</td>
<td>0.11</td>
</tr>
<tr>
<td>incremner</td>
<td>0.58</td>
</tr>
<tr>
<td><strong>VM total</strong></td>
<td><strong>0.75%</strong></td>
</tr>
<tr>
<td>OS enforcer (main)</td>
<td>0.01</td>
</tr>
<tr>
<td>OS enforcer (worker)</td>
<td>0.04</td>
</tr>
<tr>
<td>libvirtd</td>
<td>0.91</td>
</tr>
<tr>
<td><strong>QEMU</strong></td>
<td><strong>6.92</strong></td>
</tr>
<tr>
<td>VMM inspector</td>
<td>0.39</td>
</tr>
<tr>
<td><strong>VMM total</strong></td>
<td><strong>8.27%</strong></td>
</tr>
<tr>
<td>VMM enforcer</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>switch total</strong></td>
<td><strong>0.00%</strong></td>
</tr>
</tbody>
</table>

### Figure 15

<table>
<thead>
<tr>
<th></th>
<th>module ($4$)</th>
<th>spy component ($4$)</th>
<th>lines of code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>platform-independent modules</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thread in app: glue (C++)</td>
<td>app inspector</td>
<td>app inspector</td>
<td>101</td>
</tr>
<tr>
<td>thread in app: glue (Java)</td>
<td>app inspector</td>
<td>app inspector</td>
<td>241</td>
</tr>
<tr>
<td>shared enforcer code</td>
<td>all enforcers</td>
<td>client library</td>
<td>465</td>
</tr>
<tr>
<td>client library</td>
<td>client library</td>
<td>app inspector</td>
<td>1287</td>
</tr>
<tr>
<td>client library glue (Java)</td>
<td>client library</td>
<td>app inspector</td>
<td>310</td>
</tr>
<tr>
<td><strong>platform-independent total</strong></td>
<td></td>
<td></td>
<td><strong>2404</strong></td>
</tr>
<tr>
<td><strong>platform-specific modules</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app-enforcer process</td>
<td>app enforcer</td>
<td>OS inspector</td>
<td>403</td>
</tr>
<tr>
<td>incremner</td>
<td>OS inspector</td>
<td>OS inspector</td>
<td>43</td>
</tr>
<tr>
<td>kernel module</td>
<td>OS inspector</td>
<td>OS inspector</td>
<td>39</td>
</tr>
<tr>
<td>libvirtd extensions</td>
<td>OS enforcer</td>
<td>OS enforcer</td>
<td>606</td>
</tr>
<tr>
<td>OS enforcer (main)</td>
<td>OS enforcer</td>
<td>OS enforcer</td>
<td>509</td>
</tr>
<tr>
<td>OS enforcer (worker)</td>
<td>OS enforcer</td>
<td>OS enforcer</td>
<td>83</td>
</tr>
<tr>
<td>libvirtd extensions</td>
<td>OS enforcer</td>
<td>OS enforcer</td>
<td>53</td>
</tr>
<tr>
<td>RPC module</td>
<td>VMM inspector</td>
<td>VMM inspector</td>
<td>103</td>
</tr>
<tr>
<td>DD-WRT extension</td>
<td>VMM enforcer</td>
<td>VMM enforcer</td>
<td>450</td>
</tr>
<tr>
<td><strong>platform-specific total</strong></td>
<td></td>
<td></td>
<td><strong>2289</strong></td>
</tr>
<tr>
<td><strong>application-specific modules</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f() for Paxos (from [44])</td>
<td>app inspector</td>
<td>app inspector</td>
<td>17</td>
</tr>
<tr>
<td>f() for primary-backup</td>
<td>app inspector</td>
<td>app inspector</td>
<td>42</td>
</tr>
<tr>
<td>f() for ZooKeeper [31]</td>
<td>app inspector</td>
<td>app inspector</td>
<td>159</td>
</tr>
</tbody>
</table>
FUTURE

• Richer failure indication
• Monitoring across data centers
• Scalable monitoring
• Targets with multiple network interfaces
• Network failure localization
DISCUSSION – PROS/CONTRIBUTIONS

- **Falcon:** A chained network of spies monitoring different layers, uses inside information and local timeouts for fast detection, surgical killing for accuracy

- First viable and fast RFD

- Spy network

- Fast detection

- Little disruption

- Reliability

- Inexpensive
DISCUSSION – CONS/DOUBTS

• Inflexible – use of application specific knowledge (a lot)
• Safety – control access

• Testbed only includes 3 hosts connected to the same switch
• No comparison experiment conducted for disruption
• Costs? Software engineer $100,000/yr

To inject the spy component to each application specific layers?
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YOU
THANK YO