Modulo: Finding Convergence Failure Bugs in Distributed Systems with Divergence Resync Models

Beom Heyn Kim$^\dagger$, Taesoo Kim$^\dagger$, and David Lie$^\dagger$
$^\dagger$Samsung Research, $^\dagger$University of Toronto, $^\dagger$Georgia Institute of Technology
{beomheyn.kim, tsgates.kim}@samsung.com, lie@eecg.toronto.edu
Background: Various Services Rely on Replicated Distributed Storage Systems

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>OneDrive</td>
<td>Exchange Online</td>
<td>Salesforce</td>
<td>SharePoint Online</td>
<td>Yammer</td>
<td>Cisco WebEx</td>
<td>Skype for Business</td>
<td>Box</td>
<td>Zendesk</td>
<td>Oracle Taleo</td>
<td>Concur</td>
<td>Workday</td>
<td>ServiceNow</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SuccessFactors</td>
<td>Cornerstone OnDemand</td>
<td>Lithium</td>
<td>Slack</td>
<td>LastPass</td>
<td>ADP</td>
<td>DocuSign</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Replicated Distributed Storage Systems are Critical Components
Problem: A Single Bug May Cause Catastrophic Events due to an Inconsistent View of Databases

ZooKeeper / ZOOKEEPER-1319

Missing data after restarting+expanding a cluster

Details
Type: Bug
Priority: Blocker
Affects Version/s: 3.4.0

Description
The scenario I see is this:
1) Start up a 1-server ZK cluster (the server has ZK ID 0).
2) A client connects to the server, and makes a bunch of znodes, in particular a znode called "/membership".
3) Shut down the cluster.
4) Bring up a 2-server ZK cluster, including the original server 0 with its existing data, and a new server with ZK ID 1.
5) Node 0 has the highest zxid and is elected leader.
6) A client connecting to server 1 tries to "get /membership" and gets back a -101 error code (no such znode).
7) The same client then tries to "create /membership" and gets back a -110 error code (znode already exists).
8) Clients connecting to server 0 can successfully "get /membership".
Problem: A Single Bug May Cause Catastrophic Events due to an Inconsistent View of Databases

Reproduction Step:

1. Start Server 0
2. Create data items on Server 0
3. Shutdown Server 0
4. Start Server 0 and Server 1
5. Clients cannot read data items on Server 1
6. Clients can read data items on Server 0
Problem: A Single Bug May Cause Catastrophic Events due to an Inconsistent View of Databases

Reproduction Step:
1. Start Server 0
2. Create data items on Server 0
3. Shutdown Server 0
4. Start Server 0 and Server 1
5. Clients cannot read data items on Server 1
6. Clients cannot recreate the same data items, because Server 0 complains those already exist

Servers are out-of-sync!
Clients may read inconsistent data and make incorrect decisions!
Problem: A Single Bug May Cause Catastrophic Events due to an Inconsistent View of Databases

ZooKeeper / ZOOKEEPER-1319

Missing data after restarting+expanding a cluster

<table>
<thead>
<tr>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
</tr>
</tbody>
</table>

Patrick D. Hunt added a comment - 07/Dec/11 01:29

I see the problem, a change made in ZOOKEEPER-1136 is causing this regression. Setting of lastProposed in the lead() method of Leader was commented out for some reason. As a result the new follower is getting an empty diff rather than a snapshot.

This is a serious issue as it's causing the follower to get an inconsistent view of the database. We'll need to roll 3.4.1 asap.

Jeremy – Thanks for reporting this issue!
Problem: Bugs are Difficult to Find and Fix

Data inconsistency when follower is receiving a DIFF with a dirty snapshot

Description

Initial Condition
1. Let's say there are three nodes in the system.
2. The current epoch is 7.
3. For simplicity of the example, let's say that the epoch will never change.
4. The zoid is 73.
5. All the nodes have seen the change.

Step 1
Request with zoid 74 is issued. The leader A can't resolve, so A, B, and C never write the change 74 to the logs.

Step 2
A, B, and C never write the change 74 to the logs. A sends a request with zoid 74 to B, and B sends a request with zoid 74 to C.

Step 3
B, C, and A are still down.
B, C form the quorum.
B is the new leader. Let's say B's current epoch is now 80. Zoid is 80.
Request with zoid 81 is successful. On B's minCommittedItx, minCommittedItx is 81.

Step 4
A starts up. It applies the change in request with zoid 74 to its in-memory data tree.
A contacts B to registerAsFollower and provides 74 as its zoid.
Since 74 <= 74 <= 81, B decides to send A the diff.

Problems:
The problem with the above sequence is that after truncate the log, A will load the snapshot again which is not correct.
Problem: Bugs are Difficult to Find and Fix

- Created Several Sub-tasks
- Not Resolved Yet (10 Yrs)
- There are other similar bugs in Jira (recurring problem)
  → Need Automated Bug Finding Tool for These Bugs!
Replicated Distributed Storage Systems Provide Clients Consistent State/Data

Replicated Distributed Storage System

Put(1)  Client 1

Get()->1  Client 2
Key Enabler: Convergence Property Keeps Replicas Consistent

Client 1

Client 2
Step 1: A Replica Accepts Clients’ Requests

1

0

Put(1)

Client 1

Client 2
Step 2: Replicas Become Converged via Replication

Client 1

Client 2

Put(1)

Replicate

Replicate

Replicate
Step 3: Clients Read Consistent Data

Client 1

Client 2

Get() -> 1
Problem 1: What if a Replica Fails?

```
Put(1)
```

Client 1

Client 2
Step 1: A Replica Fails and Becomes Unavailable

Failures (Crash or Network)

Client 1

Client 2
Step 2: Replication Does Not Occur and Replicas Remain Diverged

Divergence across replicas remains!
Step 3: Failure Recovery and Resync Makes Replicas Converged Again

Failure Recovery (Restart or Reconnect)
Problem 2: Software Bugs in Resync Mechanisms May Cause Convergence Failures

Convergence Failure Bugs (CFBs)

Client 1

Client 2

Convergence Failure! (Never Occur)
Problems Cause Clients to Read Inconsistent Data

Client 1

Client 2

Get() -> 0
Existing Approaches: Model-based Approaches and Random Testing Approaches

• Model-Based Testing and Model-Checking
  • Problem: State space exploration is generic not targeted, therefore suffers from state explosion

• Manual Testing and Random Testing
  • Problem: state space exploration is neither systematic nor exhaustive, therefore may miss corner cases
Our Approach: Targeted, Systematic and Exhaustive State Space Exploration to Overcome Limitation of Existing Approaches

- **Model-Based Testing and Model-Checking**
  - Problem: State space exploration is generic not targeted, therefore suffers from state explosion
    - Targeted State Space Exploration

- **Manual Testing and Random Testing**
  - Problem: state space exploration is neither systematic nor exhaustive, therefore may miss corner cases
    - Systematic and Exhaustive State Space Exploration
Key Observation 1: Convergence Failure Bugs (CFB) can be Abstracted in Concise, Reproducible Steps

<table>
<thead>
<tr>
<th>Event Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash Replica A and B</td>
</tr>
<tr>
<td>Cli: Put(1)</td>
</tr>
<tr>
<td>Replica C: Processing Request</td>
</tr>
<tr>
<td>Replica C: Proposing</td>
</tr>
<tr>
<td>Replica C: Count for Txn ID</td>
</tr>
<tr>
<td>Replica C: Record in Txn log</td>
</tr>
<tr>
<td>Crash Replica C</td>
</tr>
<tr>
<td>Start Replica A and B</td>
</tr>
<tr>
<td>Replica A: Leader election begins</td>
</tr>
<tr>
<td>Replica B: Leader election begins</td>
</tr>
<tr>
<td>... (Leader election events) ...</td>
</tr>
<tr>
<td>Replica A: Following</td>
</tr>
<tr>
<td>Replica A: Getting diff from B</td>
</tr>
<tr>
<td>Replica B: LEADING</td>
</tr>
<tr>
<td>Replica B: Sync with A</td>
</tr>
<tr>
<td>Replica B: Quorum is ready</td>
</tr>
<tr>
<td>Crash Replica A and B</td>
</tr>
<tr>
<td>Put(1)</td>
</tr>
<tr>
<td>Crash Replica C</td>
</tr>
<tr>
<td>Restart Replica A and B</td>
</tr>
<tr>
<td>Resync A and B</td>
</tr>
</tbody>
</table>
Key Observation 2: Interleave Abstracted Steps to Find New CFBs

<table>
<thead>
<tr>
<th>Crash Replica A and B</th>
<th>Crash Replica A and B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cli: Put(1)</td>
<td>Put(1)</td>
</tr>
<tr>
<td>Replica C: Processing Request</td>
<td>Crash Replica C</td>
</tr>
<tr>
<td>Replica C: Proposing</td>
<td></td>
</tr>
<tr>
<td>Replica C: Count for Txn ID</td>
<td>Restart Replica A and B</td>
</tr>
<tr>
<td>Replica C: Record in Txn log</td>
<td>Resync A and B</td>
</tr>
<tr>
<td>Crash Replica C</td>
<td></td>
</tr>
<tr>
<td>Start Replica A and B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Replica A: Leader election begins</td>
<td></td>
</tr>
<tr>
<td>Replica B: Leader election begins</td>
<td></td>
</tr>
<tr>
<td>... (Leader election events) ...</td>
<td></td>
</tr>
<tr>
<td>Replica A: Following</td>
<td></td>
</tr>
<tr>
<td>Replica A: Getting diff from B</td>
<td></td>
</tr>
<tr>
<td>Replica B: LEADING</td>
<td></td>
</tr>
<tr>
<td>Replica B: Sync with A</td>
<td></td>
</tr>
<tr>
<td>Replica B: Quorum is ready</td>
<td></td>
</tr>
</tbody>
</table>
Key Idea 1: Using Divergence and Convergence Events can Even Further Reduce State Exploration

By Focusing on Interleaving Divergence and Convergence, the state space to explore is further reduced.
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)

- System-Under-Test
  - Modelling the Target Abstract Behavior
  - Modelling the Target Concrete Behavior

Divergence Resync Model (DRM)
- Abstract Execution Model (AEM)
- Concrete Execution Model (CEM)
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)

- Modelling the Target Abstract Behavior
- Modelling the Target Concrete Behavior
- Schedule Generation

Divergence Resync Model (DRM)

Abstract Execution Model (AEM)

Concrete Execution Model (CEM)

System-Under-Test
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)
Key Idea 2: Separating Abstraction from Concrete Execution (Divergence Resync Model)

Benefits of Separating Abstraction from Concrete Execution:
1. An AEM may be used for different systems-under-test
2. The common functionality of CEMs repeatedly implemented can be compiled as a library
Modulo Architecture: Schedule Generator and Concrete Executor
Modulo Architecture: Users Specify and Provide a DRM
Modulo Architecture: AEM for Schedule Generator

Modulo

Schedule Generator

Concrete Executor

AEM

CEM

DRM
Modulo Architecture: CEM for Concrete Executor
Modulo Architecture: AEM State Exploration

- **Modulo**
  - Schedule Generator
  - Concrete Executor
  - AEM State Exploration

- **System-Under-Test**

- **CEM**
  - DRM
  - AEM
  - CEM
Modulo Architecture: AEM State Exploration

Modulo

Schedule Generator

AEM

Schedule Files

Schedule 1
Schedule 2
...

Concrete Executor

CEM

System-Under-Test

AEM State Exploration

AEM

CEM

DRM

AEM

CEM

36
Modulo Architecture: CEM Input Injection

Schedule Generator

AEM

AEM State Exploration

Concrete Executor

Schedule Files

Schedule 1
Schedule 2
...

Input Generation

CEM State Exploration

Input Injection

System-Under-Test

Modulo Architecture: CEM Input Injection
Modulo Architecture: Checking if Convergence Fails after Each Schedule Execution

Get(X) -> 0
Get(Y) -> 1

Get(X) -> 0
Get(Y) -> 2

Convergence Failure Detected!
Abstract Execution Model: Each State Contains State Variables

\[
\text{replicaState} = [] \\
\text{onlineState} = []
\]
Abstract Execution Model: User-Provided Parameters
Make the State Space Concrete

numOps = 2
numReplicas = 3

A, B, C

$S_0$
replicaState = [_, _, _]
onlineState = [_, _, _]
Abstract Execution Model: Predefined Write Sequence is Generated

$$\text{setData}(X,0) \text{ setData}(X,1) \text{ setData}(X,2)$$

numOps = 2
numReplicas = 3

A,B,C

replicaState=[_,_,_]
onlineState=[_,_,_]
Abstract Execution Model: Writing Monotonically Increasing Values

numOps = 2
numReplicas = 3

A,B,C

setData(X,0) setData(X,1) setData(X,2)

Values are monotonically increasing
Abstract Execution Model: Indexing Each Write

numOps = 2
numReplicas = 3

A,B,C

setData(X,0)  setData(X,1)  setData(X,2)

Index  0  1  2

replicaState=[_,_,_]
onlineState=[_,_,_]
Abstract Execution Model: Meaning of Each State Variables

numOps = 2  
numReplicas = 3

setData(X,0)  setData(X,1)  setData(X,2)

Index 0 1 2

Up to which write
Which replicas are available

replicaState= [, , ]
onlineState= [, , ]

A, B, C

S0
Abstract Execution Model: Initial State S0

State S0

setData(X,0) → A
A: X:0

A
replicaState=[0,0,0]

onlineState=[T,T,T]

B: X:0

C: X:0
Abstract Execution Model: Applying a Divergence Transition

State S0

setData(X,0) → A

X:0

B
X:0

C
X:0

Transition T1

1 setData(X,1) → A
X:1

2

3

C
X:1

4 X:0

4 X:1

 replicaState=[0,0,0] onlineState=[T,T,T]

T1 divergence X
Abstract Execution Model: Updating State Variables

State S0

setData(X,0) → A

A:0

X:0

B:0

C:0

replicaState=[0,0,0]

onlineState=[T,T,T]

State S1

replicaState=[1,0,1]

onlineState=[F,F,F]

Transition T1

1. setData(X,1) → A

A:1

X:1

B:0

C:0

2. setData(X,1) → A

A:1

X:1

B:0

C:0

3. setData(X,1) → A

A:1

X:1

B:0

C:0

4. setData(X,1) → A

A:1

X:1

B:0

C:0
Abstract Execution Model: Applying a Convergence Transition

State S1

replicaState=[1,0,1]
onlineState=[F,F,F]

Transition T2

T2 convergence X
Abstract Execution Model: Updating State Variables

State S1
- replicaState: [1,0,1]
- onlineState: [F,F,F]

State S2
- replicaState: [1,1,1]
- onlineState: [T,T,F]

Transition T2
- T2 convergence X

S1
- replicaState: [1,0,1]
- onlineState: [F,F,F]

S2
- replicaState: [1,1,1]
- onlineState: [T,T,F]

T2 convergence X
- 1
- 2
Concrete Execution Model: Generating Inputs by Translating AEM Transitions into Concrete Test Inputs

AEM Transitions

Intermediate Representation

Concrete Test Inputs

T1 divergence X

Crash B
Write 1 to X
Crash A and C

$ kill -9 <B>
setData(X,1)
Thread.sleep(3000)
$ kill -9 <A> <C>
Concrete Execution Model: Generating Inputs by Translating AEM Transitions into Concrete Test Inputs

AEM Transitions | Intermediate Representation | Concrete Test Inputs
---|---|---
T1 divergence X | Crash B
Write 1 to X
Crash A and C | $ kill -9 <B>
setData(X,1)
Thread.sleep(3000)
$ kill -9 <A> <C>

T2 convergence X | Restart A and B
Wait for Resync | $ java ... QuorumPeerMain <A>/zoo.cfg
$ java ... QuorumPeerMain <B>/zoo.cfg
Scan logs for “LEADING” or “FOLLOWING”
Concrete Execution Model: Injecting Inputs Relative to Internal Events

<table>
<thead>
<tr>
<th>Input Injection</th>
<th>Internal Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>kill -9 &lt;B&gt;</td>
<td>A commits</td>
</tr>
<tr>
<td>setData(X,1)</td>
<td>C commits</td>
</tr>
<tr>
<td>Thread.sleep (3000)</td>
<td></td>
</tr>
<tr>
<td>kill -9 &lt;A&gt; &lt;C&gt;</td>
<td></td>
</tr>
<tr>
<td>java ... &lt;A&gt;</td>
<td>A &amp; B resyncs</td>
</tr>
<tr>
<td>java ... &lt;B&gt;</td>
<td></td>
</tr>
<tr>
<td>Scan log ...</td>
<td></td>
</tr>
<tr>
<td>Scanning Logs</td>
<td>A: “LEADING”</td>
</tr>
<tr>
<td></td>
<td>B: “FOLLOWING”</td>
</tr>
</tbody>
</table>

T1 divergence X

Crash B
Write 1 to X
Crash A and C

T2 convergence X

Restart A and B
Wait for Resync

Non-determinism Control
Implementation

• 8.4K LoC in total
  • Schedule Generator: 0.3K LoC
  • Concrete Executor: 0.8K LoC
  • Divergence Resync Models: 7.3K LoC
    • AEMs: 2.8K LoC
    • CEMs: 4.6K LoC

• Applied to 3 Replicated Distributed Storage Systems
  • ZooKeeper
  • MongoDB
  • Redis
## Modulo Found CFBs in Popular Distributed Systems

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>DRM</th>
<th>Root Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZooKeeper Bug #1 (New Bug!)</td>
<td>Q/C/Z-DRM</td>
<td>Fail to remove invalid conflicting operations (missing TRUNC invocation)</td>
</tr>
<tr>
<td>ZooKeeper Bug #2 (New Bug!)</td>
<td>Q/C/Z-DRM</td>
<td>Fail to remove invalid conflicting operations (file handling logic error)</td>
</tr>
<tr>
<td>ZooKeeper Bug #3 (New Bug!)</td>
<td>Q/C/Z-DRM</td>
<td>Fail to replicate operations due to an incomplete log</td>
</tr>
<tr>
<td>ZooKeeper Bug #4 (New Bug!)</td>
<td>Q/C/Z-DRM</td>
<td>Fail to truncate operations due to a pointer handling mistake</td>
</tr>
<tr>
<td>ZooKeeper Bug #5 (New Bug!)</td>
<td>Q/C/Z-DRM</td>
<td>Fail to truncate operations due to missing invocation</td>
</tr>
<tr>
<td>MongoDB Bug #1 (New Bug!)</td>
<td>Q/C/M-DRM</td>
<td>Fail to remove invalid conflicting operations (incomplete timestamp info)</td>
</tr>
<tr>
<td>MongoDB Bug #2</td>
<td>Q/C/M-DRM</td>
<td>Fail to replicate operations (incomplete protocol design)</td>
</tr>
</tbody>
</table>

We Found 11 CFBs:
- Newly Discovered 5 CFBs in ZooKeeper and 1 CFB in MongoDB
- Detected 1 known CFB in MongoDB and 4 known CFBs in Redis
The Size of State Space to Explore is Small Enough for Systematic and Exhaustive Search

<table>
<thead>
<tr>
<th>DRM</th>
<th>numOps</th>
<th>numReplicas</th>
<th># of Schedules</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZooKeeper’s DRM</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>1035</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>13381</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>172993</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>3428</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>54655</td>
</tr>
<tr>
<td>Redis’s DRM (Suspend)</td>
<td>2</td>
<td>4</td>
<td>13586</td>
</tr>
<tr>
<td>Redis’s DRM (Link)</td>
<td>2</td>
<td>3</td>
<td>263</td>
</tr>
<tr>
<td>Redis’s DRM (Crash+Link)</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>96</td>
</tr>
</tbody>
</table>

We could systematically and exhaustively complete state space exploration!
Separating Abstraction from Concrete Execution Makes Modulo Portable and Extensible

<table>
<thead>
<tr>
<th>DRM</th>
<th>USER/LIB</th>
<th>AEM</th>
<th>CEM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZooKeeper’s DRM</td>
<td>USER</td>
<td>54</td>
<td>59</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>LIB</td>
<td>339</td>
<td>620</td>
<td>959</td>
</tr>
<tr>
<td>MongoDB’s DRM</td>
<td>USER</td>
<td>54</td>
<td>117</td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>LIB</td>
<td>339</td>
<td>907</td>
<td>1246</td>
</tr>
<tr>
<td>Redis’s DRM (Suspend)</td>
<td>USER</td>
<td>33</td>
<td>39</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>LIB</td>
<td>955</td>
<td>1240</td>
<td>2195</td>
</tr>
<tr>
<td>Redis’s DRM (Link)</td>
<td>USER</td>
<td>0</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>LIB</td>
<td>955</td>
<td>1240</td>
<td>2195</td>
</tr>
<tr>
<td>Redis’s DRM (Crash+Link)</td>
<td>USER</td>
<td>405</td>
<td>377</td>
<td>782</td>
</tr>
<tr>
<td></td>
<td>LIB</td>
<td>955</td>
<td>1240</td>
<td>2195</td>
</tr>
</tbody>
</table>
Conclusion

• Modulo is effective in finding bugs in real-world distributed systems
  • Key Approach: Targeted, Systematic and Exhaustive State Space Exploration
  • Key Ideas
    • Exploring only interleaving of divergence and convergence
      • State space to explore is significantly reduced
    • Separating abstraction from concrete execution by decoupling them into AEM and CEM
      • Modulo becomes portable and extensible

• Modulo can be extended to find bugs in your distributed systems!
  • Github: https://github.com/Kaelus/Modulo
Thank You!

Beom Heyn Kim§†, Taesoo Kim§‡, and David Lie†
§Samsung Research, †University of Toronto, ‡Georgia Institute of Technology
{beomheyn.kim, tsgates.kim}@samsung.com, lie@eecg.toronto.edu
Abstract Execution Model: Enabled Transitions at S1

State S1

Enabled Transitions at S1

<table>
<thead>
<tr>
<th>Divergence</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>convergence A</td>
</tr>
<tr>
<td></td>
<td>convergence Z</td>
</tr>
</tbody>
</table>
Abstract Execution Model: Enabled Transitions at S0

State S0

setData(X,0) → A

B → X:0

C → X:0

replicaState = [0,0,0]

onlineState = [T,T,T]

Enabled Transitions at S0

<table>
<thead>
<tr>
<th>Divergence</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>divergence A</td>
<td></td>
</tr>
<tr>
<td>·</td>
<td>divergence Z</td>
</tr>
</tbody>
</table>
Abstract Execution Model: Initial State S0

State S0

setData(X,0) → A

replicaState=[0,0,0]
onlineState=[T,T,T]
Abstract Execution Model: Enabled Transitions at S0

State S0

setData(X,0)

Enabled Transitions at S0

<table>
<thead>
<tr>
<th>Divergence</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>divergence A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>divergence Z</td>
</tr>
</tbody>
</table>
Abstract Execution Model: Applying a Divergence Transition

State S0

setData(X,0) → A

X:0

B X:0

C X:0

Transition T1

1

setData(X,1) → A

X:1

2

3

C X:1

4

X:1

4

X:1

setData(X,0) → A

X:0

B X:0

C X:0

setReplicaState=[0,0,0]

onlineState=[T,T,T]
Abstract Execution Model: Updating State Variables

State S0

setData(X,0) → A

X:0

B X:0

C X:0

replicaState=[0,0,0]

onlineState=[T,T,T]

S0

Transition T1

1 setData(X,1) → A

X:1

B

C

X:1

T1 divergence X

2

3

4

replicaState=[1,0,1]

onlineState=[F,F,F]

State S1

setData(X,0)

X:0

X:0

X:1

X:0

X:1

X:1
Abstract Execution Model: Enabled Transitions at S1

State S1

replicaState=[1,0,1]
onlineState=[F,F,F]

Enabled Transitions at S1

<table>
<thead>
<tr>
<th>Divergence</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>convergence A</td>
</tr>
<tr>
<td></td>
<td>convergence Z</td>
</tr>
</tbody>
</table>
Abstract Execution Model: Applying a Convergence Transition

State S1

Transition T2

T2 convergence X
Abstract Execution Model: Updating State Variables

State S1

- replicaState = [1, 0, 1]
- onlineState = [F, F, F]

Transition T2

- replicaState = [1, 1, 1]
- onlineState = [T, T, F]

State S2

- replicaState = [1, 1, 1]
- onlineState = [T, T, F]
Modulo Architecture: AEM State Exploration

Modulo

Schedule Generator

Schedule Files

Schedule 1
Schedule 2
...

AEM

AEM State Exploration

Concrete Executor

DRM

AEM
CEM

System-Under-Test
Modulo Architecture CEM Input Generation

Modulo

Schedule Generator

Schedule Files

Schedule 1
Schedule 2
...

Concrete Executor

Input Generation

CEM State Exploration

AEM State Exploration

AEM

CEM

System-Under-Test

DRM

AEM

CEM

Schedule Files
Modulo Architecture: CEM Input Injection

- **Modulo**
  - Schedule Generator
  - Schedule Files
  - AEM State Exploration
  - Schedule 1
  - Schedule 2

- **Concrete Executor**
  - Input Generation
  - CEM State Exploration
  - CEM

- **DRM**
  - AEM
  - CEM

- **System-Under-Test**
  - Input Injection
Example: ZooKeeper

• Primary-backup replication
• Quorum for a leader election
• The leader serializes every write operation
• Followers replicate the write sequence directly from the leader
• After crash recovery, leader election and resync automatically begin
Example: ZooKeeper’s Divergence Resync Model

• AEM
  • Crash failures only
  • Each divergence crashes remaining online replicas at the end
  • Each convergence restarts enough number of replicas to form a quorum

• CEM
  • To Kill: $ kill -9 <A>
  • To Write: setData API call (e.g. setData(x,1))
  • To Restart: java ...QuorumPeerMain <A>/zoo.cfg
# Implementation: DRM Example Comparison

<table>
<thead>
<tr>
<th>Name</th>
<th>AEM</th>
<th>CEM</th>
<th>Lines of Code (AEM/CEM/Total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q/C/Z-DRM</td>
<td>Only consider crash failures</td>
<td>Using kill -9 for crash</td>
<td>USER 54/59/113 LIB 339/620/959</td>
</tr>
<tr>
<td></td>
<td>Convergence ensures the quorum</td>
<td>Confirm the quorum exists before writes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Crashes all replicas at the end of divergence</td>
<td>Using log scanning before 3.5, but as of 3.5, relying on timeouts</td>
<td></td>
</tr>
<tr>
<td>Q/C/M-DRM</td>
<td>Same as Q/C/Z-DRM</td>
<td>Using an API to compare timestamps of the last transaction on each replica</td>
<td>USER 54/117/171 LIB 339/907/1246</td>
</tr>
<tr>
<td></td>
<td>Considers all replicas initially partitioned</td>
<td>Using ‘info’ API and timeout to wait for resync completion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>As recovering suspend failures, establish links between the replicas</td>
<td>Using ‘slaveof’ API to trigger resync</td>
<td></td>
</tr>
<tr>
<td>S/L/R-DRM</td>
<td>Only considers link failures</td>
<td>‘slaveof’ API for link failures and recoveries. Initially, forming links as a single slave chain</td>
<td>USER 0/110/110 LIB 955/1240/2195</td>
</tr>
<tr>
<td></td>
<td>Replicas initially connected in a single chain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S/CL/R-DRM</td>
<td>Considers both link and crash failures</td>
<td>For the offline resync strategy, a script copying over snapshots and starting up a replica with the snapshot is used</td>
<td>USER 405/377/782 LIB 955/1240/2195</td>
</tr>
<tr>
<td></td>
<td>Consider two types of resync strategies: online resync and offline resync</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Schedule generation is implemented in about 281 lines of code, and concrete execution takes about 766 lines.
## Evaluation: Testing Performance

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>DRM</th>
<th>Elapsed Time</th>
<th>Time/Schedule</th>
<th># of Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZooKeeper Bug #1</td>
<td>Q/C/Z-DRM</td>
<td>11 hours</td>
<td>33 sec</td>
<td>11</td>
</tr>
<tr>
<td>ZooKeeper Bug #2</td>
<td>Q/C/Z-DRM</td>
<td>2 hours</td>
<td>39 sec</td>
<td>11</td>
</tr>
<tr>
<td>ZooKeeper Bug #3</td>
<td>Q/C/Z-DRM</td>
<td>23 min</td>
<td>33 sec</td>
<td>7</td>
</tr>
<tr>
<td>ZooKeeper Bug #4</td>
<td>Q/C/Z-DRM</td>
<td>47 min</td>
<td>30 sec</td>
<td>10</td>
</tr>
<tr>
<td>ZooKeeper Bug #5</td>
<td>Q/C/Z-DRM</td>
<td>20 hours</td>
<td>37 sec</td>
<td>10</td>
</tr>
<tr>
<td>MongoDB Bug #1</td>
<td>Q/C/M-DRM</td>
<td>18 min</td>
<td>6 min</td>
<td>3</td>
</tr>
<tr>
<td>MongoDB Bug #2</td>
<td>Q/C/M-DRM</td>
<td>4 hours</td>
<td>5 min</td>
<td>5</td>
</tr>
<tr>
<td>Redis Bug #1</td>
<td>S/S/R-DRM</td>
<td>6 hours</td>
<td>6 min</td>
<td>6</td>
</tr>
<tr>
<td>Redis Bug #2</td>
<td>S/CL/R-DRM</td>
<td>11 min</td>
<td>14 sec</td>
<td>4</td>
</tr>
<tr>
<td>Redis Bug #3</td>
<td>S/CL/R-DRM</td>
<td>2 min</td>
<td>6 sec</td>
<td>3</td>
</tr>
<tr>
<td>Redis Bug #4</td>
<td>S/L/R-DRM</td>
<td>2 min</td>
<td>33 sec</td>
<td>2</td>
</tr>
</tbody>
</table>
Conclusion

• Modulo employs targeted abstraction and concrete execution to mitigate the traditional state-explosion problems.
  • It does not explore states and state transitions that are not related to the concepts of convergence and divergence.
Abstract Execution Model: Picking a Convergence Transition

State S1

replicaState=[1,0,1]
onlineState=[F,F,F]

Enabled Transitions at S1

<table>
<thead>
<tr>
<th>Divergence</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T2 convergence [A,B]</td>
</tr>
<tr>
<td></td>
<td>convergence [B,C]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>convergence [A,B,C]</td>
</tr>
</tbody>
</table>
Abstract Execution Model: Picking a Divergence Transition

State S0

setData(X,0) → A
replicaState=[0,0,0] onlineState=[T,T,T]

Enabled Transitions at S0

<table>
<thead>
<tr>
<th>Divergence</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>divergence A</td>
<td></td>
</tr>
<tr>
<td>[0,0,1]</td>
<td></td>
</tr>
<tr>
<td>divergence [0,1,1]</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>divergence [1,0,1]</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>divergence [1,1,2]</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>divergence [2,2,2]</td>
<td></td>
</tr>
</tbody>
</table>

...
Divergence Resync Model (DRM): Specifics about the ZooKeeper DRM Example

• ZooKeeper System
  • Primary-backup replication scheme (leader and follower in ZooKeeper’s parlance)
  • Quorum is required to elect a leader
  • The leader serializes every write operation
  • Followers replicate the write sequence directly from the leader
  • After crash recovery, leader election and resync automatically begin

• DRM for ZooKeeper Specifics
  • Crash failures only
  • Each divergence crashes remaining online replicas at the end
  • Each convergence restarts enough number of replicas to form a quorum
Key Observation 1: There Exist Externally Reproducible Convergence Failure Bugs

- **Reproducing Steps:** (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B; (5) Restart A; (6) Restart C; (7) Put(k2, v2); (8) Crash A; (9) Crash C; (10) Restart B; (11) Restart C; (12) Crash B; (13) Put(k3, v3); (14) Crash C; (15) Restart B; (16) Restart C
Key Observation 1: There Exist Externally Reproducible Convergence Failure Bugs

- **Reproducing Steps:** (1) Crash A; (2) Crash C; (3) Put\((k1, v1)\); (4) Crash B; (5) Restart A; (6) Restart C; (7) Put\((k2, v2)\); (8) Crash A; (9) Crash C; (10) Restart B; (11) Restart C; (12) Crash B; (13) Put\((k3,v3)\); (14) Crash C; (15) Restart B; (16) Restart C

It will be more targeted approach to find these bugs if we explore interleaving of relevant events, e.g. Restart, Crash, Put.
Key Observation 1: There Exist Externally Reproducible Convergence Failure Bugs

- **Reproducing Steps:** (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B; (5) Restart A; (6) Restart C; (7) Put(k2, v2); (8) Crash A; (9) Crash C; (10) Restart B; (11) Restart C; (12) Crash B; (13) Put(k3,v3); (14) Crash C; (15) Restart B; (16) Restart C

It will be more targeted approach to find these bugs if we explore interleaving of relevant events, e.g. Restart, Crash, Put.

Excluding irrelevant events from state exploration
Key Observation 2: Focusing on Divergence and Convergence Further Reduces the State Space

• Reproduction Step:
  (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B;
  (5) Restart A; (6) Restart C;
  (7) Put(k2, v2); (8) Crash A; (9) Crash C;
  (10) Restart B; (11) Restart C;
  (12) Crash B; (13) Put(k3, v3); (14) Crash C;
  (15) Restart B; (16) Restart C
Key Observation 2: Focusing on Divergence and Convergence Further Reduces the State Space

• Reproduction Step:
  (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B; (5) Restart A; (6) Restart C; (7) Put(k2, v2); (8) Crash A; (9) Crash C; (10) Restart B; (11) Restart C; (12) Crash B; (13) Put(k3,v3); (14) Crash C; (15) Restart B; (16) Restart C

• Divergence and Convergence:
  (1) Divergence [0,1,0]; (2) Convergence [A,C]; (3) Divergence [1,0,1]; (4) Convergence [B,C]; (5) Divergence [0,0,1]; (6) Convergence [B,C];
Key Observation 2: Focusing on Divergence and Convergence Further Reduces the State Space

**Reproduction Step:**

1. Crash A;
2. Crash C;
3. Put\(k_1, v_1\);
4. Crash B;
5. Restart A;
6. Restart C;
7. Put\(k_2, v_2\);
8. Crash A;
9. Crash C;
10. Restart B;
11. Restart C;
12. Crash B;
13. Put\(k_3, v_3\);
14. Crash C;
15. Restart B;
16. Restart C

**Divergence and Convergence:**

1. Divergence \([0,1,0]\);
2. Convergence \([A,C]\);
3. Divergence \([1,0,1]\);
4. Convergence \([B,C]\);
5. Divergence \([0,0,1]\);
6. Convergence \([B,C]\);
Key Observation 2: Focusing on Divergence and Convergence Further Reduces the State Space

• Reproduction Step:
  (1) Crash A; (2) Crash C; (3) Put(k1, v1); (4) Crash B;
  (5) Restart A; (6) Restart C;
  (7) Put(k2, v2); (8) Crash A; (9) Crash C;
  (10) Restart B; (11) Restart C;
  (12) Crash B; (13) Put(k3, v3); (14) Crash C;
  (15) Restart B; (16) Restart C

• Divergence and Convergence:
  (1) Divergence [0,1,0];
  (2) Convergence [A,C];
  (3) Divergence [1,0,1];
  (4) Convergence [B,C];
  (5) Divergence [0,0,1];
  (6) Convergence [B,C];

We can reduce a sequence of low level events into a sequence of higher level divergence and convergence events.
Related Works: Exhaustive State Search Suffers from State Explosion

• Model-based testing (OAuthTester, MBTC) and model-checking (PACE, CMC, Verisoft, MaceMC, MODIST, CrystalBall, dBug, SAMC, FlyMC, etc.): employing state-space exploration to systematically check for the absence of bugs
  • Limitation: state space exploration is usually generic and not targeted, therefore suffers from the state explosion
Related Works: Non-Systematic State Search May Miss Bugs

- Model-based testing (OAuthTester, MBTC) and model-checking (PACE, CMC, Verisof, MaceMC, MODIST, CrystalBall, dBug, SAMC, FlyMC, etc.): employing state-space exploration to systematically check for the absence of bugs
  - Limitation: state space exploration is usually generic and not targeted, therefore suffers from the state explosion
- Manual testing and random testing (Jepsen): Scope of testing is usually targeted to find specific types of bugs
Related Works: Non-Systematic State Search May Miss Bugs

- Model-based testing (OAuthTester, MBTC) and model-checking (PACE, CMC, Verisoft, MaceMC, MODIST, CrystalBall, dBug, SAMC, FlyMC, etc.): employing state-space exploration to systematically check for the absence of bugs
  - Limitation: state space exploration is usually generic and not targeted, therefore suffers from the state explosion

- Manual testing and random testing (Jepsen): Scope of testing is usually targeted to find specific types of bugs
  - Limitation: state space exploration is neither systematic nor exhaustive, therefore may miss corner cases
Background: Convergence Property Keeps Replicas Consistent

Client 1

Put(X,1)

Client 2
Background: Convergence Property Keeps Replicas Consistent
Background: Convergence Property Keeps Replicas Consistent
Background: Divergence Can Be Observed by Clients

X=1
Put(X,1)

X=0

X=0

Client 1

Client 2
Background: Divergence Can Be Observed by Clients

Client 1

Replicate

Client 2
Background: Divergence Can Be Observed by Clients

Client 1

Client 2

Get(X) -> 0
Background: Failures Extends Divergence’s Lifetime Until Recovery and Resync

Failures (Crash or Network)
Background: Recovery and Resync Reduces Divergence and Restores Convergence

Failure Recovery (Restart or Reconnect)
Background: Recovery and Resync Removes Divergence and Restores Convergence
Background: Software Bugs in Resync Mechanisms May Cause Convergence Failures

Convergence Failure Bugs (CFBs)
Divergence and Convergence

Replicated Distributed Storage Systems

Operation Requests

Client Applications
Divergence and Convergence

Replicated Distributed Storage Systems

Client Applications
Divergence and Convergence

Replicated Distributed Storage Systems

Client Applications
Divergence and Convergence

**Divergence**: A process that replicas become different

**Convergence**: A process that replicas become equivalent

Replicated Distributed Storage Systems

Client Applications
Convergence Property

Replicated Distributed Storage Systems

Convergence Property

Client Applications
Consistency Models

Replicated Distributed Storage Systems

Convergence Failures

Client Applications
Convergence Failure Bugs (CFBs) Can Occur

Client

Put(X, 0)

Distributed storage

A

B

C

X: 0

X: 0

X: 0
Convergence Failure Bugs (CFBs) Can Occur

Divergence

Crash or Connection Failures

Distributed storage
Convergence Failure Bugs (CFBs) Can Occur

1. Client
   - Put(X,0)

2. A
   - X:0
   - Divergence

3. B
   - X:0

4. B
   - X:0
   - Crash or Connection Failures

5. Put(X,1)

6. A
   - X:1

7. C
   - X:1
   - Convergence

8. B
   - X:0
   - Distributed storage

Convergence or Crash or Connection Failures
Convergence Failure Bugs (CFBs) Can Occur

Distributed storage

Convergence

Crash or Connection Failures
Convergence Failure Bugs (CFBs) Can Occur

- **Divergence**
- **Convergence**
- **Crash or Connection Failures**

**Client**

1. **Put(X,0)**
   - **A** X:0
   - **B** X:0
   - **C** X:0

2. **Put(X,0)**
   - **A** X:0
   - **B** X:0
   - **C** X:0

3. **Put(X,1)**
   - **A** X:1
   - **B** X:0
   - **C** X:1

4. **Put(X,1)**
   - **A** X:1
   - **B** X:0
   - **C** X:1

5. **Put(X,1)**
   - **A** X:1
   - **B** X:0
   - **C** X:1

6. **Put(X,1)**
   - **A** X:1
   - **B** X:0
   - **C** X:1

7. **Put(X,1)**
   - **A** X:1
   - **B** X:0
   - **C** X:1

8. **Put(X,1)**
   - **A** X:1
   - **B** X:0
   - **C** X:1

9. **Put(X,1)**
   - **A** X:1
   - **B** X:0
   - **C** X:1

10. **Put(X,1)**
    - **A** X:1
    - **B** X:0
    - **C** X:1

Distributed storage
Convergence Failure Bugs (CFBs) Can Occur

Divergence

1. Client
2. Put(X,0)
3. B
4. X:0

Convergence

5. Put(X,1)
6. A
7. A
8. B
9. Convergence Failure Bugs (CFBs)
10. Convergence Failure!

Crash or Connection Failures

Convergence Failures → Incorrect Decisions of Client Apps

Goal: Finding Convergence Failure Bugs!
Limitations of Existing Techniques

• Model-based testing and model-checking: employing state-space exploration to systematically check for the absence of bugs
  • Limitation: state space exploration is usually generic and not targeted, therefore suffers from the state explosion

• Manual testing and random testing: Scope of testing is usually targeted to find specific types of bugs
  • Limitation: state space exploration is neither systematic nor exhaustive, therefore may miss corner cases

Modulo: Using a targeted approach to abstraction and concrete execution based on that abstraction to overcome those limitations
Data Consistency?

Replicated Distributed Storage Systems

Wait for Replication?
Just show stale data?
How much stale?

Client Applications
Data Consistency?

Replicated Distributed Storage Systems

Strict Ordering?
How to determine the order?

Replicated Distributed Storage Systems

Client Applications
Consistency Models

Replicated Distributed Storage Systems

Client Applications

Consistency Models

Timing

Order
Consistency Models

Replicated Distributed Storage Systems

Consistency Models

Convergence Property

Client Applications
Model-based Testing with Divergence Resync Models

System-Under-Test (SUT)

Divergence Resync Model (DRM)
Model-based Testing with Divergence Resync Models

Abstract Execution Model (AEM)

Concrete Execution Model (CEM)

Divergence Resync Model (DRM)

Modelling the Target Abstract Behavior

Modelling the Target Concrete Behavior

System-Under-Test (SUT)
Model-based Testing with Divergence Resync Models

Abstract Execution Model (AEM)

Concrete Execution Model (CEM)

Divergence Resync Model (DRM)

System-Under-Test (SUT)

Modelling the Target Abstract Behavior

Modelling the Target Concrete Behavior

Modelling the Target Abstract Behavior
Model-based Testing with Divergence Resync Models

Divergence Resync Model (DRM)

Abstract Execution Model (AEM)

Concrete Execution Model (CEM)

Modelling the Target Abstract Behavior

Modelling the Target Concrete Behavior

Schedule Generation
Model-based Testing with Divergence Resync Models

Abstract Execution Model (AEM)

Concrete Execution Model (CEM)

Divergence Resync Model (DRM)

Modelling the Target Abstract Behavior

Modelling the Target Concrete Behavior

Schedule Generation

Input Generation
Differences in DRMs

• Q/C/Z-DRM CEM
  • Before version 3.5, scanning log to see each replica switches their roles after leader election to wait for the resync completion
  • Since version 3.5, log scanning is no longer reliable, thus fall back to time delay
Differences in DRMs

• Q/C/Z-DRM CEM
  • Before version 3.5, scanning log to see each replica switches their roles after leader election to wait for the resync completion
  • Since version 3.5, log scanning is no longer reliable, thus fall back to time delay

• Q/C/M-DRM
  • For MongoDB, but AEM is same as Q/C/Z-DRM
  • For CEM, it uses an API to get timestamps of the last transaction on each replica to confirm that resync completes
Differences in DRMs

• **Q/C/Z-DRM CEM**
  • Before version 3.5, scanning log to see each replica switches their roles after leader election to wait for the resync completion
  • Since version 3.5, log scanning is no longer reliable, thus fall back to time delay

• **Q/C/M-DRM**
  • For MongoDB, but AEM is same as Q/C/Z-DRM
  • For CEM, it uses an API to get timestamps of the last transaction on each replica to confirm that resync completes

• **S/S/R-DRM, S/L/R-DRM, S/CL/R-DRM**
  • Models for Redis uses more failure models, including link failures which requires extended AEM to keep track the status of network links between replicas
Divergence Resync Model (DRM): Differences in DRMs

• DRM for ZooKeeper
  • Before version 3.5, scanning log to see each replica switches their roles after leader election to wait for the resync completion
  • Since version 3.5, log scanning is no longer reliable, thus fall back to time delay

• DRM for MongoDB
  • For MongoDB, but AEM is same as the DRM for ZooKeeper
  • For CEM, it uses an API to get timestamps of the last transaction on each replica to confirm that resync completes

• DRMs for Redis
  • Models for Redis uses more failure models, including link failures which requires extended AEM to keep track the status of network links between replicas
Modulo Architecture

Schedule Generator

Concrete Executor

Divergence: [1,0,1]

Convergence: [A,B]

Schedule Files

Verification Result
A.X = B.X = C.X

Modulo Architecture Diagram:

- **Schedule Generator**
  - AEM
  - Schedule Files
  - Schedule 1
  - Schedule 2

- **Concrete Executor**
  - Divergence: [1,0,1]
  - Convergence: [A,B]

- **CEM**
  - CEM State Exploration

- **System-Under-Test**
  - $\text{kill -9 <B>}$
  - $\text{setData(X,1)}$
  - $\text{Thread.sleep(3000)}$
  - $\text{kill -9 <A> <C>}$

- **CEM State Exploration**

- **DRM**

- **AEM**

- **AEM State Exploration**

Code Snippets:

```java
$ java ... <A>/zoo.cfg
$ java ... <B>/zoo.cfg
Scan logs ...
```
## State Space Size

<table>
<thead>
<tr>
<th>DRM</th>
<th>numOps</th>
<th>numReplicas</th>
<th># of Schedules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q/C/Z</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>1035</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>13381</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>172993</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>3428</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>54655</td>
</tr>
<tr>
<td>S/S/R</td>
<td>2</td>
<td>4</td>
<td>13586</td>
</tr>
<tr>
<td>S/L/R</td>
<td>2</td>
<td>3</td>
<td>263</td>
</tr>
<tr>
<td>S/CL/R</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>96</td>
</tr>
</tbody>
</table>
Discussion

• Methodology
  • First, write DRMs in a top-down approach
  • Second, focus on the specific behavior that is important to manifest target bugs
  • Third, pay attention to configuration parameters of the system-under-test
• Modulo requires users manual effort to provide DRMs
  • Target users are developers with expertise who are interested in stress the specific behavior of the system-under-test.
  • For novice users, we expect that it requires about 2 weeks to learn about the system-under-test and about 2 weeks to write DRMs
  • Effective DRMs do require a good intuition and insight about target bugs
Conclusion

• Modulo employs targeted abstraction and concrete execution to mitigate the traditional state-explosion problems.
  • It does not explore states and state transitions that are not related to the concepts of convergence and divergence.

• Our work identified several factors that lead to CFBs:
  • (1) employing several resync or failure-handling mechanisms whose interactions are difficult to foresee
  • (2) hard limits or inadequate designs for handling large amounts of divergence
  • (3) assumptions about length of time that replicas may have failed and failures that span events like leader transitions.

• Modulo’s performance is heavily affected to delays from executing and controlling the real distributed system