CPSC 426/526
Large-Scale Failure Prevention

Ennan Zhai
Computer Science Department
Yale University
Recall: Lec-16

• In lec-16, we learned:
  - Configuration failures in software
  - How to handle configuration errors
  - Case study: Spex and ConfigV
Lecture Roadmap

• Why cloud-scale failures occur
• Why they are challenging
• Case Study: INDaaS
• Case Study: RepAudit
Reliability via Redundancy

• Cloud services ensure reliability by redundancy:
Reliability via Redundancy

- Cloud services ensure reliability by redundancy:
  - Amazon S3 replicates data on multiple racks
Cloud services ensure reliability by redundancy:
- Amazon S3 replicates data on multiple racks
- iCloud rents EC2 and Azure redundantly
Reliability via Redundancy

• Cloud services ensure reliability by redundancy:
  - Amazon S3 replicates data on multiple racks
  - iCloud rents EC2 and Azure redundantly

Does redundancy really help?
Example
Example

Rack 1

Rack 2

Rack 3

Switch 1

Switch 2

Switch 3
Example
Example

Rack 1
- Replica
- Switch 1

Rack 2
- Replica
- Switch 2

Rack 3
- Replica
- Switch 3

Agg Switch
Example

Agg Switch

Rack 1
Replica
Switch 1

Rack 2
Replica
Switch 2

Rack 3
Replica
Switch 3

Replica

Example
Example
Unexpected correlated failures

Switch1

Replica

Replica

Replica

Switch2

Agg Switch

Switch3
Correlated Failures in Real-World
Correlated Failures in Real-World

We’d like to share more about the service event that occurred on Monday, October 22nd in the US-East Region. We have now completed the analysis of the events that affected AWS customers, and we want to describe what happened, our understanding of how customers were affected, and what we are doing to prevent a similar issue from occurring in the future.

The Primary Event and the Impact to Amazon Elastic Block Store (EBS) and Amazon Elastic Compute Cloud (EC2)
Amazon EC2/EBS Stack

Elastic Compute Cloud (EC2)

Elastic Block Store (EBS)
Amazon EC2/EBS Stack

Elastic Block Store (EBS)
Amazon EC2/EBS Stack

Elastic Block Store (EBS)
Amazon EC2/EBS Stack

VM1  VM2  VM3  VM4

Elastic Block Store (EBS)
Amazon EC2/EBS Stack

VM1
VM2
VM3
VM4

EBS Server1
EBS Server2
Amazon is not alone
Amazon is not alone

• In Google redundant systems:
  - 37% failures are correlated;
  - each of them needs many hours to fix [1].

• In HBase, correlated failures may disable whole clusters
  - 34 logic bugs;
  - 13 bugs in error-handling modules [2].

Even Worse
Lightning strikes Amazon's European cloud

**Summary:** The lightning strike damaged a power company's transformer, causing disruption to Amazon Web Services's European cloud, and may have affected Microsoft's BPOS as well.

The outage, which Amazon Web Services (AWS) acknowledged on Sunday evening, affected its Dublin-based Elastic Compute Cloud (EC2) and Relational Database Service (RDS) cloud services, among others. The damage to the electricity infrastructure may have affected Microsoft's Business Productivity Online Services (BPOS) cloud as well, Microsoft said in a separate statement.
Video App

Cloud Provider A

Cloud Provider B

Third-party infrastructure components
Cloud providers do not usually share information about their dependencies.
State-of-the-Arts
State-of-the-Arts

- Cloud providers allocate or tolerate failures via:
  - diagnosis systems;
  - fault-tolerant systems.
State-of-the-Arts

- Cloud providers allocate or tolerate failures via:
  - diagnosis systems;
  - fault-tolerant systems.
- Solving the problem after outage occurs.
State-of-the-Arts

• Cloud providers allocate or tolerate failures via:
  - diagnosis systems;
  - fault-tolerant systems.

• Solving the problem after outage occurs.

• Very time-consuming for today’s complex systems.
State-of-the-Arts

• Cloud providers allocate or tolerate failures via:
  - diagnosis systems;
  - fault-tolerant systems.

• Solving the problem after outage occurs.

In Google, Matt Welsh’s case:

The engineer on call spent many, many hours trying different things and trying to isolate the problem, without success.
State-of-the-Arts

- Cloud providers allocate or tolerate failures via:
  - diagnosis systems;
  - fault-tolerant systems.

- Solving the problem after outage occurs.

- Very time-consuming for today’s complex systems.
- Cannot prevent service downtime
Cloud providers allocate or tolerate failures via:

- diagnosis systems;
- fault-tolerant systems.

Solving the problem after outage occurs.

Very time-consuming for today’s complex systems.

Cannot prevent service downtime.

Disease prevention is better than diagnosis

— World Health Organization

Cannot prevent service downtime
Failure Prevention

Service Provider, Alice

Prevention System

A Given Redundancy Configuration
Two-Way Redundancy Configuration

Amazon S3

Service Provider, Alice

Prevention System

Data Source1

Data Source2
Two-Way Redundancy Configuration

Service Provider, Alice

Prevention System

Two-Way Redundancy Configuration

EC2
Data Source 1

Azure
Data Source 2
Failure Prevention

Service Provider, Alice

Independence of this two-way redundancy?

Prevention System

Two-Way Redundancy Configuration

Dependency Data Source 1

Dependency Data Source 2
Failure Prevention

Service Provider, Alice

Step 1: Specification
Submission

Prevention System

Dependency
Data Source 1

Dependency
Data Source 2
Failure Prevention

Service Provider, Alice

Step 1

Prevention System

Step 2: Dependency data collection

Dependency Data Source 1

Step 2: Dependency data collection

Dependency Data Source 2
Failure Prevention

Service Provider, Alice

Step 1

Prevention System

Step 2

Dependency Data Source 1

Step 2

Dependency Data Source 2

Step 3: Independence Evaluation
Failure Prevention

Service Provider, Alice

Prevention System

Dependency Data Source 1

Relative Independence is 0.3

Switch 1 is a dangerous source

Step 1

Step 2

Step 3
Failure Prevention in Multi-Cloud

Service Provider, Alice

Step 1

Prevention System

Step 2

Dependency Data Source 1

Step 2

Dependency Data Source 2

Step 3
Failure Prevention in Multi-Cloud

Service Provider, Alice

Prevention System

Data Source1 (Cloud1)

Data Source2 (Cloud2)

Step 1

Step 2

Step 3

Unwilling to share the dependency data

Unwilling to share the dependency data
Failure Prevention in Multi-Cloud

Service Provider, Alice

Step 1

Prevention System

Step 2

Data Source 1 (Cloud 1)

Data Source 2 (Cloud 2)

Step 3: Private auditing
Failure Prevention in Multi-Cloud

Service Provider, Alice

Step 1

Prevention System

Step 2

Data Source 1 (Cloud 1)

Step 4

Data Source 2 (Cloud 2)

Step 3
Failure Prevention in Multi-Cloud

Service Provider, Alice

Data Source1 (Cloud1)

Data Source2 (Cloud2)

Only know my information

Step 1

Prevention System

Step 2

Step 3

Step 4
Failure Prevention in Multi-Cloud

Service Provider, Alice

Only the relative independence

Prevention System

Step 1

Step 2

Step 4

Data Source 1 (Cloud 1)

Step 3

Data Source 2 (Cloud 2)
Failure Prevention in Multi-Cloud

Service Provider, Alice

Prevention System

Data Source1 (Cloud1)  Data Source2 (Cloud2)

Step1  Step2  Step4  Step2  Step4

Relative Independence is 0.3
Lecture Roadmap

• Why cloud-scale failures occur
• Why they are challenging
• Case Study: INDaaS
• Case Study: RepAudit
Challenges

Prevention System

Step 1

Step 2

Step 4

Dependency Data Source 1

Dependency Data Source 2

Step 3

Step 5
Challenges

- #1: Dependency collections
  - Solution: Reusing existing tools
Challenges

- #1: Dependency collections
  - Solution: Reusing existing tools

- #2: Dependency representation
  - Solution: Fault graphs
Challenges

• #1: Dependency collections
  - Solution: Reusing existing tools

• #2: Dependency representation
  - Solution: Fault graphs

• #3: Efficient auditing
  - Solution: Failure sampling algorithm
Challenges

- **#1: Dependency collections**
  - Solution: Reusing existing tools
- **#2: Dependency representation**
  - Solution: Fault graphs
- **#3: Efficient auditing**
  - Solution: Failure sampling algorithm
- **#4: Private independence audit**
  - Solution: Private Jaccard similarity
Challenges

• #1: Dependency collections
  - Solution: Reusing existing tools

• #2: Dependency representation
  - Solution: Fault graphs

• #3: Efficient auditing
  - Solution: Failure sampling algorithm

• #4: Private independence audit
  - Solution: Private Jaccard similarity
Challenges

- **#1: Dependency collections**
  - Solution: Reusing existing tools

- **#2: Dependency representation**
  - Solution: Fault graphs

- **#3: Efficient auditing**
  - Solution: Failure sampling algorithm

- **#4: Private independence audit**
  - Solution: Private Jaccard similarity
Lecture Roadmap

• Why cloud-scale failures occur
• Why they are challenging
• Case Study: INDaaS
• Case Study: RepAudit
Dependency Data Collections

• Different clouds have various topologies
• Outputs of the collection tools are different
Dependency Data Collections

• Reuse existing data collection tools:
  - Convert the outputs to uniform format.
  - Three types of format: NET, HW and SW.

Our defined format

<table>
<thead>
<tr>
<th>Type</th>
<th>Dependency Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>&lt;src=&quot;S&quot; dst=&quot;D&quot; route=&quot;x,y,z&quot;/&gt;</td>
</tr>
<tr>
<td>Hardware</td>
<td>&lt;hw=&quot;H&quot; type=&quot;T&quot; dep=&quot;x&quot;/&gt;</td>
</tr>
<tr>
<td>Software</td>
<td>&lt;pgm=&quot;S&quot; hw=&quot;H&quot; dep=&quot;x,y,z&quot;/&gt;</td>
</tr>
</tbody>
</table>
Dependency Data Collections
Dependency Data Collections

NSDMiner

Diagram:
- ToR Switch1 (ToR1)
- S1
- CPU1
- Disk1
- S2
- CPU2
- Disk2
- Core Router1 (Core1)
- Core Router2 (Core2)
- Internet
NSDMiner

Dependency Data Collections

Diagram:
- NSDMiner
- S1
- S2
- ToR Switch1 (ToR1)
- Core Router1 (Core1)
- Core Router2 (Core2)
- Internet

Components:
- CPU1
- Disk1
- CPU2
- Disk2
<src="S1" dst="Internet" route="ToR1,Core1"/>
<src="S1" dst="Internet" route="ToR1,Core2"/>
<src="S2" dst="Internet" route="ToR1,Core1"/>
<src="S2" dst="Internet" route="ToR1,Core2"/>

NSDMiner
<src="S1" dst="Internet" route="ToR1,Core1"/>
<src="S1" dst="Internet" route="ToR1,Core2"/>
<src="S2" dst="Internet" route="ToR1,Core1"/>
<src="S2" dst="Internet" route="ToR1,Core2"/>

NSDMiner
Example

NSDMiner

DepDB

Core Router1
(Core1)

Core Router2
(Core2)

ToR Switch1
(ToR1)

CPU1
Disk1

S1

S2

CPU2
Disk2

Internet
Challenges

• #1: Dependency collections
  - Solution: Reusing existing tools

• #2: Dependency representation
  - Solution: Fault graphs

• #3: Efficient auditing
  - Solution: Failure sampling algorithm

• #4: Private independence audit
  - Solution: Private Jaccard similarity
**Challenges**

1. Dependency collections
   - Solution: Reusing existing tools
2. Dependency representation
   - Solution: Fault graphs
3. Efficient auditing
   - Solution: Failure sampling algorithm
4. Private independence audit
   - Solution: Private Jaccard similarity

---

**INDaasS**

Step 1: Dependency Data Source 1

Step 2: Dependency Data Source 2

Step 3: Reusing existing tools

Step 4: Fault graphs

Step 5: Failure sampling algorithm

Step 6: Private Jaccard similarity
Fault Graph

- Root fails
  - E1 fails
    - A1 fails
  - E2 fails
    - A2 fails
    - A3 fails

Fault Graph

- Root fails
  - AND gate: output fault occurs if all of the input faults occur
  - E1 fails
    - OR gate: output fault occurs if at least one of the input faults occur
      - A1 fails
      - A2 fails
  - E2 fails
    - A2 fails
    - A3 fails
Example
Example
Building a fault graph
Step 1: Root Node

Redundancy configuration fails
Step 2: Server Nodes

Redundancy configuration fails

- Server 1 fails
- Server 2 fails
Step 2: Server Nodes

Redundancy configuration fails

Server 1 fails

Server 2 fails

AND gate: all the sublayer nodes fail, the upper layer node fails
Step3: Dependency Nodes

Redundancy configuration fails

Server 1 fails
- HW fails
- Net fails
- SW fails

Server 2 fails
- Net fails
- SW fails
- HW fails
Step 3: Dependency Nodes

Redundancy configuration fails

Server 1 fails

Server 2 fails

HW fails

Net fails

SW fails

Net fails

SW fails

HW fails

OR gate: one of the sublayer nodes fails, the upper layer node fails
Step 4: Hardware Dependency

Redundancy configuration fails

Server 1 fails

- HW fails
- Net fails
- SW fails

CPU1

Disk1

Server 2 fails

- Net fails
- SW fails
- HW fails

CPU2

Disk2
Step 5: Network Dependency

**Redundancy configuration fails**

- **Server 1 fails**
  - **HW fails**
  - **Net fails**
  - **SW fails**
    - CPU1
    - Disk1
    - Path1
    - Core1
    - ToR1
  - Path2
  - Core2

- **Server 2 fails**
  - **Net fails**
  - **SW fails**
  - **HW fails**
    - CPU2
    - Disk2
Step6: Software Dependency

Redundancy configuration fails

- Server 1 fails
  - HW fails
    - CPU1
    - Disk1
  - Net fails
    - Path1
    - ToR1
  - SW fails
    - Core1
    - Core2
- Server 2 fails
  - Net fails
    - Path2
    - ToR1
  - SW fails
    - Core2
  - HW fails
    - CPU2
    - Disk2
Step 6: Software Dependency

The process is totally automatic, i.e., without instruments.
Challenges

• #1: Dependency collections
  - Solution: Reusing existing tools

• #2: Dependency representation
  - Solution: Fault graphs

• #3: Efficient auditing
  - Solution: Failure sampling algorithm

• #4: Private independence audit
  - Solution: Private Jaccard similarity
Challenges

• #1: Dependency collections
  - Solution: Reusing existing tools

• #2: Dependency representation
  - Solution: Fault graphs

• #3: Efficient auditing
  - Solution: Failure sampling algorithm

• #4: Private independence audit
  - Solution: Private Jaccard similarity
A risk group means a set of leaf nodes whose simultaneous failures lead to the failure of root node.
Risk Groups in Fault Graph

A risk group means a set of leaf nodes whose simultaneous failures lead to the failure of root node.

\{A2\} and \{A1, A3\} are risk groups
\{A1\} or \{A3\} is not risk group
Risk Groups in Fault Graph

Identifying shared dependencies is reduced to the problem of finding risk groups in the fault graph.

{A2} and {A1, A3} are risk groups
{A1} or {A3} is not a risk group
Finding risk groups by analyzing fault graph

- Two algorithms balancing cost and accuracy:
  - Minimal cut set algorithm
  - Failure sampling algorithm
Finding risk groups by analyzing fault graph

- Two algorithms balancing cost and accuracy:
  - Minimal cut set algorithm
  - Failure sampling algorithm
Minimal Cut Set Algorithm

Service fails

Server 1 fails

Server 1's hw fails

Switch 1 fails

Server 2 fails

Switch 1 fails

Server2's hw fails
Minimal Cut Set Algorithm

Service fails

- Server 1 fails
  - Server 1's hardware fails
  - Switch 1 fails
- Server 2 fails
  - Switch 1 fails
  - Server 2's hardware fails

First step
Minimal Cut Set Algorithm

Step 1
Service fails
Minimal Cut Set Algorithm

Step 1

Service fails

Second step

- Server 1 fails
  - Server 1’s hardware fails
  - Switch 1 fails
- Server 2 fails
  - Switch 1 fails
  - Server 2’s hardware fails
Step 1
Service fails

Step 2
S1 fails & S2 fails

Minimal Cut Set Algorithm
Minimal Cut Set Algorithm

Service fails

Server 1 fails

- Server 1’s hardware fails
- Switch 1 fails

Server 2 fails

- Switch 1 fails
- Server 2’s hardware fails

Third step

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service fails</td>
<td>S1 fails &amp; S2 fails</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 1: Service fails

- Server 1 fails
  - Server 1’s hardware fails
  - Switch 1 fails
- Server 2 fails
  - Switch 1 fails
  - Server 2’s hardware fails

Fourth step: 

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service fails</td>
<td>S1 fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; S2 fails</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Switch 1 fails &amp; S2 fails</td>
</tr>
</tbody>
</table>
Minimal Cut Set Algorithm

- **Step 1**: Service fails
- **Step 2**: Server 1 fails & Server 2 fails
  - Server 1’s hardware fails
  - Switch 1 fails
  - Switch 1 fails
  - Server 2’s hardware fails
- **Step 3**: S1’s hw fails & S2 fails
  - S1’s hw fails & Switch 1 fails
  - Switch 1 fails & S2 fails
  - S1’s hw fails & Switch 2’s hw fails
- **Step 4**: Switch 1 fails & Switch 1 fails
  - Switch 1 fails & S2’s hw fails

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service fails</td>
<td>S1 fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; Switch 1 fails</td>
</tr>
<tr>
<td></td>
<td>Switch 1 fails</td>
<td>Switch 1 fails</td>
<td>Switch 1 fails &amp; S2 fails</td>
</tr>
<tr>
<td></td>
<td>Switch 1 fails</td>
<td>Switch 2’s hw fails</td>
<td>Switch 1 fails &amp; Switch 1 fails</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Switch 1 fails &amp; S2’s hw fails</td>
<td></td>
</tr>
</tbody>
</table>
Minimal Cut Set Algorithm

- **Step 1**: Service fails
- **Step 2**: Server 1 fails & Server 2 fails
  - **Step 3**: Server 1’s hw fails & Switch 1 fails
    - **Step 4**: Switch 1 fails & Switch 1 fails
  - **Step 3**: Switch 1 fails & Switch 1 fails
    - **Step 4**: Switch 1 fails & S2’s hw fails
  - **Step 3**: Server 2’s hw fails
    - **Step 4**: Switch 1 fails & S2’s hw fails
Minimal Cut Set Algorithm

- Service fails
  - Server 1 fails
    - Server 1’s hardware fails
    - Switch 1 fails
  - Server 2 fails
    - Server 2’s hardware fails
    - Switch 1 fails

- Simplify them

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service fails</td>
<td>S1 fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; Switch1 fails</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Switch1 fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Switch1 fails &amp; Switch1 fails</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Switch1 fails &amp; S2’s hw fails</td>
<td></td>
</tr>
</tbody>
</table>
Minimal Cut Set Algorithm

Service fails

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
<td>S1’s hw fails &amp; Switch1 fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
</tr>
<tr>
<td>S1 fails</td>
<td>Switch1 fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
<td>Switch1 fails &amp; Switch1 fails</td>
</tr>
<tr>
<td>Server 1 fails</td>
<td>Switch1 fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
<td>Switch1 fails &amp; S2’s hw fails</td>
</tr>
<tr>
<td>Server 1’s hw</td>
<td>Switch1’s hw fails &amp; S2’s hw fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
<td>Switch1 fails &amp; S2’s hw fails</td>
</tr>
<tr>
<td>hardware fails</td>
<td>Switch1 fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
<td>Switch1 fails &amp; S2’s hw fails</td>
</tr>
<tr>
<td>Switch 1 fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
<td>Switch1 fails &amp; S2’s hw fails</td>
</tr>
<tr>
<td>Switch 2 fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
<td>Switch1 fails &amp; S2’s hw fails</td>
</tr>
</tbody>
</table>
Minimal Cut Set Algorithm

Service fails

Server 1 fails
- Server 1’s hardware fails
- Switch 1 fails

Server 2 fails
- Switch 1 fails
- Server 2’s hardware fails

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service fails</td>
<td>S1 fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; Switch 1 fails</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Switch 1 fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Switch 1 fails</td>
<td>Switch 1 fails</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Switch 1 fails &amp; S2’s hw fails</td>
</tr>
</tbody>
</table>
Minimal Cut Set Algorithm

- **Step 1**: Service fails
  - **Step 2**: Server 1 fails
    - **Step 3**: Server 1’s hardware fails
    - **Step 4**: Server 1’s hardware fails & Switch 1 fails
  - **Step 2**: Switch 1 fails
  - **Step 2**: Server 2 fails
    - **Step 3**: Server 2’s hardware fails
    - **Step 4**: Server 1’s hardware fails & Switch 1 fails

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service fails</td>
<td>S1 fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; Switch 1 fails</td>
</tr>
<tr>
<td></td>
<td>Switch 1 fails</td>
<td>Switch 1 fails &amp; S2 fails</td>
<td>Switch 1 fails</td>
</tr>
<tr>
<td></td>
<td>Switch 1 fails</td>
<td>Switch 1 fails &amp; S2’s hw fails</td>
<td>Switch 1 fails &amp; S2’s hw fails</td>
</tr>
<tr>
<td></td>
<td>Server 2’s hardware fails</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Minimal Cut Set Algorithm

Two minimal risk groups:

\{Switch1 fails\} and 
\{Server1’s hw fails & Server2’s hw fails\}

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service fails</td>
<td>S1 fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; S2 fails</td>
<td>S1’s hw fails &amp; S2’s hw fails</td>
</tr>
<tr>
<td></td>
<td>Switch1 fails</td>
<td>Switch1 fails</td>
<td>Switch1 fails</td>
</tr>
<tr>
<td></td>
<td>Server1’s hw fails</td>
<td>Server2’s hw fails</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Minimal Cut Set Algorithm

• Traditional algorithm in safety engineering
  - Exponential complexity (NP-hard)

• We are the first to apply it in Cloud area:
  - Analyzing a fat tree with 30,528 with ~40 hours
Minimal Cut Set Algorithm

- Traditional algorithm in safety engineering
  - Exponential complexity (NP-hard)

- We are the first to apply it in Cloud area:
  - Analyzing a fat tree with 30,528 with ~40 hours

- We propose efficient failure sampling algorithm.
Fault Graph Analysis

- Two algorithms balancing cost and accuracy:
  - Minimal cut set algorithm
  - Failure sampling algorithm
Redundancy configuration fails

Server 1 fails

Server 1’s HW fails

Server 2 fails

Switch 1 fails

Server2’s HW fails
Failure Sampling Algorithm

Redundancy configuration fails

- Server 1 fails
  - Server 1’s HW fails
    - 1 or 0

- Server 2 fails
  - Switch 1 fails
    - 1 or 0
  - Server 2’s HW fails
    - 1 or 0
Failure Sampling Algorithm

Redundancy configuration fails

Server 1 fails

Server 1’s HW fails

Switch 1 fails

Switch 1’s HW fails

Server 2 fails

Server 2’s HW fails

1 or 0?
Fault Sets

Redundancy configuration fails

Server 1 fails

Server 1’s HW fails

Switch 1 fails

Server 2 fails

Server 2’s HW fails

Failure Sampling Algorithm

Fault Sets

∅
The 1st Sampling Round

Redundancy configuration fails

- Server 1 fails
  - Server 1’s HW fails
- Server 2 fails
  - Server 2’s HW fails

Fault Sets

∅
The 1st Sampling Round

Redundancy configuration fails

Server 1 fails

Server 1’s HW fails

1 or 0

Server 2 fails

Switch 1 fails

1 or 0

Server 2’s HW fails

1 or 0

Fault Sets

∅
The 1st Sampling Round

Redundancy configuration fails

Server 1 fails
- Server 1’s HW fails

Server 2 fails
- Switch 1 fails
- Server 2’s HW fails

Fault Sets

∅
The 1st Sampling Round

Redundancy configuration fails

Server 1 fails
Server 2 fails

Server 1’s HW fails
Switch 1 fails
Server 2’s HW fails

Fault Sets
∅
The 1st Sampling Round

Fault Sets

{Server1’s HW, Server2’s HW}
The 2nd Sampling Round

Redundancy configuration fails

Server 1 fails

Server 1's HW fails

Switch 1 fails

Server 2 fails

Server 2's HW fails

Fault Sets

\{Server 1's HW, Server 2's HW\}
The 2nd Sampling Round

Redundancy configuration fails

- Server 1 fails
  - Server 1’s HW fails
    - 1 or 0
  - Switch 1 fails
    - 1 or 0
- Server 2 fails
  - Server 2’s HW fails
    - 1 or 0

Fault Sets

{Server 1’s HW, Server 2’s HW}
The 2nd Sampling Round

Redundancy configuration fails

- Server 1 fails
  - Server 1’s HW fails
  - Switch 1 fails

- Server 2 fails
  - Server 2’s HW fails

Fault Sets

\{Server 1’s HW, Server 2’s HW\}
The 2nd Sampling Round

Redundancy configuration fails

- Server 1 fails
- Server 2 fails

Server 1’s HW fails
- Switch 1 fails
- Server 2’s HW fails

Fault Sets

{Server 1’s HW, Server 2’s HW}
The 3rd Sampling Round

Fault Sets

{Server1’s HW, Server2’s HW}
The 3rd Sampling Round

Redundancy configuration fails

- Server 1 fails
- Server 2 fails

- Server 1’s HW fails
- Switch 1 fails
- Server 2’s HW fails

Fault Sets

\{Server 1’s HW, Server 2’s HW\}
The 3rd Sampling Round

Redundancy configuration fails

- Server 1 fails
- Server 2 fails
- Server1’s HW fails
- Server2’s HW fails
- Switch1 fails

Fault Sets

{Server1’s HW, Server2’s HW}
The 3rd Sampling Round

Fault Sets

- \{\text{Server 1's HW, Server 2's HW}\}
- \{\text{Switch 1}\}
After Many (e.g., $10^7$) Rounds

<table>
<thead>
<tr>
<th>Fault Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Server1’s HW, Server2’s HW}</td>
</tr>
<tr>
<td>{Switch1}</td>
</tr>
<tr>
<td>{Switch1, Server2’s HW}</td>
</tr>
<tr>
<td>{Switch1}</td>
</tr>
<tr>
<td>{Switch1, Server2’s HW}</td>
</tr>
</tbody>
</table>

... ...
## Size-Based Ranking

<table>
<thead>
<tr>
<th>Fault Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Server1’s HW, Server2’s HW}</td>
</tr>
<tr>
<td>{Switch1}</td>
</tr>
<tr>
<td>{Switch1, Server2’s HW}</td>
</tr>
<tr>
<td>{Switch1}</td>
</tr>
<tr>
<td>{Switch1, Server2’s HW}</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
## Size-Based Ranking

<table>
<thead>
<tr>
<th>Fault Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Switch1}</td>
</tr>
<tr>
<td>{Switch1}</td>
</tr>
<tr>
<td>{Switch1}</td>
</tr>
<tr>
<td>{Switch1, Server2’s HW}</td>
</tr>
<tr>
<td>{Switch1, Server2’s HW}</td>
</tr>
<tr>
<td>{Server1’s HW, Server2’s HW}</td>
</tr>
</tbody>
</table>
Independence Evaluation

- Multiple equations for option:
  - summation of sizes
  - weighted average of sizes

<table>
<thead>
<tr>
<th>Fault Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Switch1}</td>
</tr>
<tr>
<td>{Switch1}</td>
</tr>
<tr>
<td>{Switch1, Server2’s HW}</td>
</tr>
<tr>
<td>{Switch1, Server2’s HW}</td>
</tr>
<tr>
<td>{Server1’s HW, Server2’s HW}</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
We can do better if we can obtain failure probabilities
Using Failure Probabilities

Redundancy configuration fails

Server 1 fails

Server 1's HW fails

0.2

Server 2 fails

Switch 1 fails

Server 2's HW fails

0.3

0.1
Using Failure Probabilities

Redundancy configuration fails

- Server 1 fails
  - Server 1’s HW fails
    - 0.2
- Server 2 fails
  - Switch 1 fails
    - 0.3
  - Server 2’s HW fails
    - 0.1

Fault Sets

<table>
<thead>
<tr>
<th>Fault Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Server 1’s HW, Server 2’s HW}</td>
</tr>
<tr>
<td>{Switch 1}</td>
</tr>
</tbody>
</table>
Using Failure Probabilities

\[
F(\{\text{Server1's HW, Server2's HW}\}) = 0.2 \times 0.1
\]

Fault Sets

<table>
<thead>
<tr>
<th>Fault Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Server1's HW, Server2's HW}</td>
</tr>
<tr>
<td>{Switch1}</td>
</tr>
</tbody>
</table>
Using Failure Probabilities

\[ F(\{\text{Server1’s HW, Server2’s HW}\}) = 0.2 \times 0.1 \]

\[ F(\{\text{Switch1}\}) = 0.3 \]

\[ F(R) = 0.3 + (0.2 \times 0.1) - 0.3 \times 0.2 \times 0.1 = 0.314 \]
Using Failure Probabilities

\[
\begin{align*}
F(\{\text{Server1’s HW, Server2’s HW}\}) &= 0.2 \times 0.1 \\
F(\{\text{Switch1}\}) &= 0.3 \\
F(R) &= 0.3 + (0.2 \times 0.1) - 0.3 \times 0.2 \times 0.1 = 0.314
\end{align*}
\]

Fault Sets

<table>
<thead>
<tr>
<th>Fault Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Server1’s HW, Server2’s HW}</td>
</tr>
<tr>
<td>{Switch1}</td>
</tr>
</tbody>
</table>

0.2

0.3

0.1
Challenges

- **#1: Dependency collections**
  - Solution: Reusing existing tools

- **#2: Dependency representation**
  - Solution: Fault graphs

- **#3: Efficient auditing**
  - Solution: Failure sampling algorithm

- **#4: Private independence audit**
  - Solution: Private Jaccard similarity
Challenges

• #1: Dependency collections
  - Solution: Reusing existing tools

• #2: Dependency representation
  - Solution: Fault graphs

• #3: Efficient auditing
  - Solution: Failure sampling algorithm

• #4: Private independence audit
  - Solution: Private Jaccard similarity
Challenges

- **#1: Dependency collections**
  - Solution: Reusing existing tools

- **#2: Dependency representation**
  - Solution: Fault graphs

- **#3: Efficient auditing**
  - Solution: Failure sampling algorithm

- **#4: Private independence audit**
  - Solution: Private Jaccard similarity
Service Provider

INDaaS Agent

Cloud A

Cloud B

Cloud C
Cloud A

Service Provider

Cloud B

INDaaS Agent

Cloud C

ISP A

Power A

ISP B

Power B

Power C

Select two clouds for redundancy: A&B? B&C? or A&C?
Select two clouds for redundancy: A&B? B&C? or A&C?

Service Provider

Cloud A
ISP A, Power A

Cloud B
ISP B, Power B

Cloud C
ISP B, Power C

INDaaS Agent

Trusted Third Party
Cloud providers are reluctant to share this information! Select two clouds for redundancy: A&B? B&C? or A&C?
Secure Multiparty Computation (SMPC)

Select two clouds for redundancy: A&B? B&C? or A&C?

Service Provider

INDaaS Agent

Cloud A

Cloud B

Cloud C

ISP A  Power A

Power B

ISP B

Power C
Secure Multiparty Computation (SMPC)

Select two clouds for redundancy: A&B? B&C? or A&C?

SMPC is hard to scale!
[Xiao et al. CCSW’13]
Select two clouds for redundancy: A&B? B&C? or A&C?

Service Provider

INDaaS Agent

Cloud A

Cloud B

Cloud C

ISP A  Power A  Power B  ISP B  Power C
Evaluating independence by the dataset similarity between clouds

Service Provider

INDaaS Agent

Cloud A

ISP A  Power A

Cloud B

ISP B

Power B

Cloud C

ISP C  Power C

Power A  ISP A

ISP B  Power B

ISP C  Power C
Jaccard similarity ->
the independence of redundancy configuration
\[
J(S_1, S_2, \ldots, S_n) = \frac{|S_1 \cap S_2 \cap \ldots \cap S_n|}{|S_1 \cup S_2 \cup \ldots \cup S_n|}
\]
Service Provider

ISP A
Power A
Power B

Cloud A

ISP B
Power A
Power B

Cloud B

ISP B
Power C

Cloud C

INDaaS Agent

ISP A
Power A

ISP B
Power B

ISP B
Power C

ISP B
Power C

Service Provider

ISP A
Power A

ISP B
Power B

ISP B
Power C

ISP B
Power C
Cloud A
ISP A
Power A
Power B

Cloud B
ISP B
Power A
Power B

Cloud C
ISP B
Power C

ISP A
Power A
Power B
Power C

Service Provider

INDaaS Agent

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud A&amp;B</td>
<td>0.5</td>
</tr>
</tbody>
</table>

\[ J = \frac{2}{4} \]
Service Provider

ISP A
Power A
Power B

Cloud A

ISP B
Power A
Power B

Cloud B

ISP B
Power B

Cloud C

INDaaS Agent

Deployment | Sim
---|---
Cloud A&B | 0.5
Cloud B&C | 0.25

\[ J = \frac{1}{4} \]
\[ \cap = 0, \quad \cup = 5, \quad j = 0/5 \]

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud A&amp;B</td>
<td>0.5</td>
</tr>
<tr>
<td>Cloud B&amp;C</td>
<td>0.25</td>
</tr>
<tr>
<td>Cloud A&amp;C</td>
<td>0</td>
</tr>
</tbody>
</table>
Cloud A
ISP A
Power A
Power B

Cloud B

Cloud C
ISP B
Power C

0 means fully independent

\[ \cap = 0, \quad \cup = 5, \quad J = 0/5 \]

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud A&amp;B</td>
<td>0.5</td>
</tr>
<tr>
<td>Cloud B&amp;C</td>
<td>0.25</td>
</tr>
<tr>
<td>Cloud A&amp;C</td>
<td>0</td>
</tr>
</tbody>
</table>

Service Provider

DaaS Agent

ISP A
Power A

ISP B

ISP C
Service Provider

ISP A
Power A

Cloud A

ISP B
Power C

Cloud B

ISP B
Power C

Cloud C

ISP A
Power A

ISP B

Cloud B & Cloud C

0.5

Cloud B & Cloud C

0.25

Cloud A & Cloud C

0

\[ \bigcap = 0, \quad \bigcup = 5, \quad J = 0/5 \]

INDaaS Agent

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud A&amp;B</td>
<td>0.5</td>
</tr>
<tr>
<td>Cloud B&amp;C</td>
<td>0.25</td>
</tr>
<tr>
<td>Cloud A&amp;C</td>
<td>0</td>
</tr>
</tbody>
</table>

Service Provider

ISP A
Power A

ISP B
Power C

ISP A

ISP B

ISP C

Power A

Power B

Power C
Service Provider

ISP A
Power A
Power B

Cloud A

∪ = 0,

Cloud B

ISP B
Power C

Cloud C

ISP B

\[ J = 0/5 \]

INDaaS Agent

Cloud A&C

Cloud B&C

Cloud A&B

0

0.25

0.5

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud A&amp;C</td>
<td>0</td>
</tr>
<tr>
<td>Cloud B&amp;C</td>
<td>0.25</td>
</tr>
<tr>
<td>Cloud A&amp;B</td>
<td>0.5</td>
</tr>
</tbody>
</table>
P-SOP [Vaidya et al. JCS05]

- We apply Private Set Operation Protocol (P-SOP):
  - Private set intersection cardinality.
  - Private set union cardinality.

\[
J(S_1, S_2, ..., S_n) = \frac{|S_1 \cap S_2 \cap ... \cap S_n|}{|S_1 \cup S_2 \cup ... \cup S_n|}
\]
P-SOP [Vaidya et al. JCS05]

- Allow k parties to compute both intersection and union cardinalities without learning other information.
P-SOP [Vaidya et al. JCS05]

• Allow k parties to compute both intersection and union cardinalities without learning other information.
P-SOP [Vaidya et al. JCS05]

- Allow $k$ parties to compute both intersection and union cardinalities without learning other information.
P-SOP [Vaidya et al. JCS05]

- Allow k parties to compute both intersection and union cardinalities without learning other information.

$\bigcap = 1, \bigcup = 7$
P-SOP [Vaidya et al. JCS05]

- Allow k parties to compute both intersection and union cardinalities without learning other information.
Each party maintains a commutative encryption key

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key,

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key

Commutative encryption holds: \( E_x(E_y(m)) = E_y(E_x(m)) \)
Each party maintains a commutative encryption key
Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key
Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key.

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key
Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key

Commutative encryption holds: \( E_x(E_y(m)) = E_y(E_x(m)) \)
Each party maintains a commutative encryption key
Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key:

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key

Commutative encryption holds: $E_x(E_y(m)) = E_y(E_x(m))$
Each party maintains a commutative encryption key:

\[ E_x(E_y(m)) = E_y(E_x(m)) \]
Each party maintains a commutative encryption key

Commutative encryption holds: \( E_x(E_y(m)) = E_y(E_x(m)) \)
Private Independence Evaluation
Select two clouds for redundancy: A&B? B&C? or A&C?

Service Provider

INDaaS Agent

Cloud A

Cloud B

Cloud C

ISP A  Power A

Power B

ISP B

Power C
Cloud A
ISP A
Power A
Power B

Cloud B
ISP B
Power A
Power B

Cloud C
ISP B
Power C

Service Provider

INDaaS Agent

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud A&amp;B</td>
<td>0.5</td>
</tr>
<tr>
<td>Cloud B&amp;C</td>
<td>0.25</td>
</tr>
</tbody>
</table>

\[ J = \frac{1}{4} \]
\[ \bigcap = 0, \quad \bigcup = 5, \quad J = 0/5 \]
Cloud A
ISP A
Power A
Power B

Cloud B

Cloud C

ISP B
Power C

ISP A
Power A
Power B

ISP B

ISP C

INDaaS Agent

\[ \cap = 0, \quad \cup = 5, \quad J = 0/5 \]

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Sim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud A&amp;B</td>
<td>0.5</td>
</tr>
<tr>
<td>Cloud B&amp;C</td>
<td>0.25</td>
</tr>
<tr>
<td>Cloud A&amp;C</td>
<td>0</td>
</tr>
</tbody>
</table>

Service Provider
\[ \bigcap = 0, \quad \bigcup = 5, \quad J = 0/5 \]
Service Provider

ISP A
Power A

ISP B
Power C

ISP A
Power B

ISP B
Power A

ISP B
Power C

ISP A
Power A

ISP B
Power B

ISP C
Power C

Deployment
Cloud A&C
0

Cloud B&C
0.25

Cloud A&B
0.5

\[ \cap = 0, \quad \cup = 5, \quad J = 0/5 \]
More Discussions

• Parties might be malicious in practice:
  - Zero-knowledge proof can solve this problem
More Discussions

• Parties might be malicious in practice:
  - Zero-knowledge proof can solve this problem

• Malicious detection (e.g., using one element):
  - Set a threshold (say, > 1000 elements) to avoid malicious detection behavior.
More Discussions

• Parties might be malicious in practice:
  - Zero-knowledge proof can solve this problem

• Malicious detection (e.g., using one element):
  - Set a threshold (say, > 1000 elements) to avoid malicious detection behavior.

• Can we make it differentially private:
  - A differential privacy set intersection cardinality protocol
Challenges

- **#1: Dependency collections**
  - Solution: Reusing existing tools

- **#2: Dependency representation**
  - Solution: Fault graphs

- **#3: Efficient auditing**
  - Solution: Failure sampling algorithm

- **#4: Private independence audit**
  - Solution: Private Jaccard similarity

INDaaS Agent

Step 1

Step 2

Step 3

Step 4

Step 5
Lecture Roadmap

• Why cloud-scale failures occur
• Why they are challenging
• Case Study: INDaaS
• Case Study: RepAudit
Service initialization → Service Runtime
Service initialization

Post-Failure Forensics
1. Diagnosis tools
2. Accountability
3. Provenance
4. .......
Service initialization

INDaaS [OSDI’14]

Post-Failure Forensics
1. Diagnosis tools
2. Accountability
3. Provenance
4. ... ...

Service Runtime
INDaaS [OSDI’14]

• INDaaS does pre-deployment recommendations:
INDaaS [OSDI’14]

- INDaaS does pre-deployment recommendations:
  - Step1: Automatically collecting dependency data
INDaaS [OSDI’14]

- INDaaS does pre-deployment recommendations:
  - Step1: Automatically collecting dependency data
  - Step2: Modeling system stack in fault graph
  - Step3: Evaluating independence of alternative redundancy configurations
Issues in INDaaS

- Hard to express diverse auditing tasks, e.g., identifying risks
Issues in INDaaS

• Hard to express diverse auditing tasks, e.g., identifying risks

• Fault graph analysis does not support auditing in runtime
Issues in INDaaS

- Hard to express diverse auditing tasks, e.g., identifying risks
- Fault graph analysis does not support auditing in runtime
Issues in INDaaS

• Hard to express diverse auditing tasks, e.g., identifying risks

• Fault graph analysis does not support auditing in runtime

• Have no idea how to fix the cascading failure problem
Issues in INDaaS

- Hard to express diverse auditing tasks, e.g., identifying risks
- Fault graph analysis does not support auditing in runtime
- Much faster analysis based on various SAT solvers
- Have no idea how to fix the cascading failure problem

Our Solution:
RepAudit [OOPSLA’17]

Changes in INDaaS

- Service initialization
- Changing network paths
- Upgrading software components

Service Runtime
RepAudit

- Hard to express diverse auditing tasks, e.g., identifying risks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime

- Have no idea how to fix the cascading failure problem

![Service initialization](circle-green)
![Changing network paths](circle-red)
![Upgrading software components](circle-purple)

---

**Service Runtime**
RepAudit

- Hard to express diverse auditing tasks, e.g., identifying risks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime
  - Much faster analysis based on various SAT solvers
- Have no idea how to fix the cascading failure problem
**RepAudit**

- Hard to express diverse auditing tasks, e.g., identifying risks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime
  - Much faster analysis based on various SAT solvers
- Have no idea how to fix the cascading failure problem
  - Automatically generate improvement plans

![Service Runtime Diagram](image-url)
RepAudit Example
RepAudit Example

\[ CNF = (A_2) \land (A_1 \lor A_3) \]

- let Server("172.28.228.21") -> s1
- let Server("172.28.228.22") -> s2
- let [s1, s2] -> rep
- let FaultGraph(rep) -> ft
- let RankRCG(ft, 2, NET, ft) -> ranklist

1. {Core1["75.142.33.98"]}
2. {Agg1["10.0.0.1"], Agg2["10.0.0.2"]}

RepAudit Example
\[ CNF = (A_2 \land (A_1 \lor A_3)) \]

Let:
- Server("172.28.228.21") -> s1
- Server("172.28.228.22") -> s2
- \[s1, s2\] -> rep
- FaultGraph(rep) -> ft
- RankRCG(ft, 2, NET, ft) -> ranklist

1. \{Core1\["75.142.33.98"\]\}
2. \{Agg1\["10.0.0.1"\], Agg2\["10.0.0.2"\]\}

Core Router 1
(Core1)

Core Router 2
(Core2)

Agg Switch 3
(Agg3)

Server 2 (S2)
172.28.228.22

Server 3 (S3)
172.28.228.23

Internet

Service Deployment
(network/software stacks)
CNF = (A₂) ˄ (A₁ ˅ A₃)

let Server("172.28.228.21") -> s₁
let Server("172.28.228.22") -> s₂
let [s₁, s₂] -> rep
let FaultGraph(rep) -> ft
let RankRCG(ft, 2, NET, ft) -> ranklist

Auditing Program

1. {Core1["75.142.33.98"]}
2. {Agg1["10.0.0.1"], Agg2["10.0.0.2"]}

Service Deployment (network/software stacks)
\[ \text{CNF} = (A_2 \land (A_1 \lor A_3)) \]

- let Server("172.28.228.21") -> s1
- let Server("172.28.228.22") -> s2
- let [s1, s2] -> rep
- let FaultGraph(rep) -> ft
- let RankRCG(ft, 2, NET, ft) -> ranklist

**Auditing Program**

1. {Core1["75.142.33.98"]}
2. {Agg1["10.0.0.1"], Agg2["10.0.0.2"]}

**Service Deployment**
(network/software stacks)

**Auditing Engine**

- Replication
- Replica1
- Replica2
- A1
- A2
- A3

**INDaas**
data collection

- HBase
- HDFS

**Internet**

- Core Router1 (Core1)
- Core Router2 (Core2)
- Agg Switch1 (Agg1)
- Agg Switch2 (Agg2)
- Agg Switch3 (Agg3)

**Server**

- Server1 (S1) 172.28.228.21
- Server2 (S2) 172.28.228.22
- Server3 (S3) 172.28.228.23
- Server4 (S4) 172.28.228.24

**HDFS**
let Server("172.28.228.21") -> s1
let Server("172.28.228.22") -> s2
let [s1, s2] -> rep
let FaultGraph(rep) -> ft
let RankRCG(ft, 2, NET, ft) -> ranklist

CNF = (A2) \& (A1 \lor A3)
let Server("172.28.228.21") -> s1
let Server("172.28.228.22") -> s2
let [s1, s2] -> rep
let FaultGraph(rep) -> ft
let RankRCG(ft, 2, NET, ft) -> ranklist

1. {Core1["75.142.33.98"]}
2. {Agg1["10.0.0.1"], Agg2["10.0.0.2"]}

CNF = (A2 ˄ (A1 ˅ A3))
RepAudit Contributions

Auditing Program

let Server("172.28.228.21") -> s1
let Server("172.28.228.22") -> s2
let [s1, s2] -> rep
let FaultGraph(rep) -> ft
let RankRCG(ft, 2, NET, ft) -> ranklist

Auditing Results

1. {Core1["75.142.33.98"]}
2. {Agg1["10.0.0.1"], Agg2["10.0.0.2"]}

Auditing Engine

INDaaS data collection

Replication

Replica1 Replica2

A1 A2 A3

<Weight Vector>

Weighted MaxSAT solver

CNF = (A2) \land (A1 \lor A3)

Service Deployment (network/software stacks)
RepAudit Contributions

Auditing Program

let Server("172.28.228.21") \to s1
let Server("172.28.228.22") \to s2
let \{s1, s2\} \to rep
let FaultGraph(rep) \to ft
let RankRCG(ft, 2, NET, ft) \to ranklist

Auditing Results

1. \{Core1["75.142.33.98"]\}
2. \{Agg1["10.0.0.1"], Agg2["10.0.0.2"]\}

CNF = (A2) \land (A1 \lor A3)

<Weight Vector>

Weighted MaxSAT solver

Auditing Engine

INDaaS data collection

Replication

Replica1

Replica2

A1

A2

A3

HBase

HDFS

HBase

HDFS

HBase

HDFS

Core Router1 (Core1)

Core Router2 (Core2)

Agg Switch1 (Agg1)

Agg Switch2 (Agg2)

Agg Switch3 (Agg3)

Server1 (S1) 172.28.228.21

Server2 (S2) 172.28.228.22

Server3 (S3) 172.28.228.23

Server4 (S4) 172.28.228.24

10.0.0.3

Internet

Service Deployment (network/software stacks)

Auditing Program in RAL

Weighted MaxSAT solver

Replication

Replica Contributions
Auditing Language

\[
S ::= \text{let } e \rightarrow g \text{ Assignment} \\
| \quad \text{print}(e) \quad \text{Output} \\
| \quad S_1;S_2 \mid \text{if}(e)\{S_1\} \text{ else}\{S_2\} \mid \text{while}(e)\{S\}
\]

(a) Statements of RAL.

\[
e ::= g \mid c \mid l(e) \mid q \mid e_1 \text{ op } e_2 \quad \text{Expression}
\]
\[
c ::= i \mid \text{str} \quad \text{Real number or string}
\]
\[
l(e) ::= \text{nil} \mid [e_1, \ldots, e_n] \quad \text{List}
\]
\[
op ::= < \mid \leq \mid = \mid != \mid > \mid \geq \quad \text{Operator}
\]
\[
q ::= \text{Server}(e) \quad \text{Initializing server node}
\]
\[
| \quad \text{Switch}(e) \quad \text{Initializing switch node}
\]
\[
| \quad \text{FaultGraph}(e) \quad \text{Generating fault graph}
\]
\[
| \quad \text{RankRCG}(e_1,e_2,m,t) \quad \text{Ranking RCGs}
\]
\[
| \quad \text{RankNode}(e,m,t) \quad \text{Ranking devices}
\]
\[
| \quad \text{FailProb}(e,t) \quad \text{Failure probability}
\]
\[
| \quad \text{RecRep}(e_1,e_2,m) \quad \text{Recommendation}
\]
\[
| \quad \ldots
\]
\[
m ::= \text{SIZE} \mid \text{PROB} \quad \text{Ranking metric}
\]
\[
t ::= \text{NET} \mid \text{SoftW} \mid \text{HardW} \quad \text{Dependency types}
\]

(b) Expressions of RAL.
Auditng Language

\[
S ::= \text{let } e \rightarrow g \quad \text{Assignment} \\
| \quad \text{print}(e) \quad \text{Output} \\
| \quad S_1;S_2 \mid \text{if}(e)\{S_1\} \text{ else } \{S_2\} \mid \text{while}(e)\{S\}
\]
(a) Statements of RAL.

\[
e ::= g \mid c \mid l(e) \mid q \mid e_1 \ op \ e_2 \quad \text{Expression}
\]
\[
c ::= i \mid \text{str} \quad \text{Real number or string}
\]
\[
l(e) ::= \text{nil} \mid [e_1, \ldots, e_n] \quad \text{List}
\]
\[
op ::= < \mid \leq \mid = \mid != \mid > \mid \geq \quad \text{Operator}
\]
\[
q ::= \text{Server}(e) \quad \text{Initializing server node}
\]
\[
| \quad \text{Switch}(e) \quad \text{Initializing switch node}
\]
\[
| \quad \text{FaultGraph}(e) \quad \text{Generating fault graph}
\]
\[
| \quad \text{RankRCG}(e_1,e_2,m,t) \quad \text{Ranking RCGs}
\]
\[
| \quad \text{RankNode}(e,m,t) \quad \text{Ranking devices}
\]
\[
| \quad \text{FailProb}(e,t) \quad \text{Failure probability}
\]
\[
| \quad \text{RecRep}(e_1,e_2,m) \quad \text{Recommendation}
\]
\[
| \quad \ldots
\]
\[
m ::= \text{SIZE} \mid \text{PROB} \quad \text{Ranking metric}
\]
\[
t ::= \text{NET} \mid \text{SoftW} \mid \text{HardW} \quad \text{Dependency types}
\]
(b) Expressions of RAL.
let s1 = Server("172.28.228.21");
let s2 = Server("172.28.228.22");
let rep = s1::s2::nil;
let ft = FaultGraph(rep);
let list = RankRCG(ft, 2, SIZE);
print(list);

1. {Core-Router-1["75.142.33.98"]}
2. {Agg-Switch-1["10.0.0.1"], Agg-Switch-2["10.0.0.2"]}
**RepAudit Contributions**

**Auditing Program**
- let Server(“172.28.228.21”) -> s1
- let Server(“172.28.228.22”) -> s2
- let [s1, s2] -> rep
- let FaultGraph(rep) -> ft
- let RankRCG(ft, 2, NET, ft) -> ranklist

**Auditing Results**
1. {Core1[“75.142.33.98”]}
2. {Agg1[“10.0.0.1”], Agg2[“10.0.0.2”]}

**CNF** = (A2) ∧ (A1 v A3)

<Weight Vector>

**Auditing Engine**

**Replication**
- Replica1
- Replica2
- A1, A2, A3

**Weighted MaxSAT solver**

**INDaaS data collection**

**Service Deployment** (network/software stacks)

Server1 (S1) 172.28.228.21
Server2 (S2) 172.28.228.22
Server3 (S3) 172.28.228.23
Server4 (S4) 172.28.228.24

Agg Switch1 (Agg1) 10.0.0.1
Agg Switch2 (Agg2) 10.0.0.2
Agg Switch3 (Agg3) 10.0.0.3

Core Router1 (Core1) 75.142.33.98
Core Router2 (Core2) 75.142.33.99

Internet
Recall: Risk Groups
A risk group means a set of leaf nodes whose simultaneous failures lead to the failure of root node.
A risk group means a set of leaf nodes whose simultaneous failures lead to the failure of root node.

\{A2\} and \{A1, A3\} are risk groups
\{A1\} or \{A3\} is not risk group
A risk group means a set of leaf nodes whose simultaneous failures lead to the failure of root node. 

\{A2\} and \{A1, A3\} are risk groups

\{A1\} or \{A3\} is not risk group
Recall: Risk Group Analysis

- State-of-the-art risk group detection efforts:
  - Deterministic minimal cut set algorithm
  - Failure sampling algorithm
Recall: Risk Group Analysis

- State-of-the-art risk group detection efforts:
  - Deterministic minimal cut set algorithm
  - Failure sampling algorithm

Pros: 100% accurate results
Cons: Exponential-time complexity
Recall: Risk Group Analysis

- State-of-the-art risk group detection efforts:
  - Deterministic minimal cut set algorithm
  - Failure sampling algorithm

Pros: 100% accurate results
Cons: Exponential-time complexity

Pros: Efficient auditing approach
Cons: Accuracy is quite low in large system
Recall: Risk Group Analysis

- State-of-the-art risk group detection efforts:
  - Deterministic minimal cut set algorithm
  - Failure sampling algorithm

We want to achieve both efficiency and accuracy in large-scale system auditing

Pros: Efficient auditing approach
Cons: Accuracy is quite low in large system
Our Insight

Boolean formula

\[ E_1 \land E_2 \]

\[ = (A_1 \lor A_2) \land (A_2 \lor A_3) \]
Boolean formula
\[ E_1 \land E_2 \]
\[ = (A_1 \lor A_2) \land (A_2 \lor A_3) \]

Our Insight

- Extracting risk groups can be reduced to the problem of extracting satisfying assignments from boolean formula

- E.g., \{A1=0, A2=1, A3=0\} represents a risk group
Our Insight

Boolean formula
= \( E_1 \land E_2 \)
= \((A_1 \lor A_2) \land (A_2 \lor A_3)\)

{A1=0, A2=1, A3=0}

SAT solver
Our Insight

- Problem:
  - Standard SAT solver outputs an arbitrary satisfying assignment
  - What we want is top-k minimal risk groups

{A1=0, A2=1, A3=0}
Extracting Risk Groups
Extracting Risk Groups

- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy
Extracting Risk Groups

- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy

We set the values of all the leaf nodes as 1
Extracting Risk Groups

- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy

![Redundancy Deployment Diagram]

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Extracting Risk Groups

- Using weighted MaxSAT solver
  - Satisfiable assignment with the least weights
  - Obtain the least $C = \sum c_i \cdot w_i$
  - Very fast with 100% accuracy
Using weighted MaxSAT solver
- Satisfiable assignment with the least weights
- Obtain the least \( C = \sum c_i \cdot w_i \)
- Very fast with 100% accuracy

Extracting Risk Groups

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Extracting Risk Groups

- Find out the top-k critical risk groups
  - Use a $\land$ to connect the current formula and negation of the resulting assignment
Extracting Risk Groups

- Find out the top-k critical risk groups
  - Use a $\land$ to connect the current formula and negation of the resulting assignment

$$(A_1 \lor A_2) \land (A_2 \lor A_3) \land \neg(A_1 \land A_2 \land \neg A_3)$$
If we can obtain failure probability of each component, then

<table>
<thead>
<tr>
<th></th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0.3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td>0.03</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>0.02</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td>0.06</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.006</td>
</tr>
</tbody>
</table>
If we can obtain failure probability of each component, then

<table>
<thead>
<tr>
<th></th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.03</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.03</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.006</td>
</tr>
</tbody>
</table>

Redundancy Deployment

Data Source E1

Data Source E2

0.1  0.3  0.2
Failure Probability Computation
Failure Probability Computation

Redundancy Deployment

Data Source E1
A1
A2
A3
0.1

Data Source E2

0.2
0.15
let Server("172.28.228.21") -> s1;
let Server("172.28.228.22") -> s2;
let [s1, s2] -> rep;
let FaultGraph(rep) -> ft;
let FailProb(ft, NET) -> prob;
print(prob);
let Server("172.28.228.21") -> s1;
let Server("172.28.228.22") -> s2;
let [s1. s2] -> rep;
let FaultGraph(rep) -> ft;
let FailProb(ft, NET) -> prob;
print(prob);

Failure Probability Computation

CNF formula \[\text{Model Counter} \rightarrow \text{The \# of satisfying assignments}\]
Model Counter
Model Counter

\[(A_1 \lor A_2) \land (A_2 \lor A_3)\]
Model Counter

\[(A_1 \lor A_2) \land (A_2 \lor A_3)\]
Model Counter

\[(A_1 \lor A_2) \land (A_2 \lor A_3)\]

<table>
<thead>
<tr>
<th></th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

5
Model Counter

If we assume the failure probability of each leaf node is 0.5

\[(A_1 \lor A_2) \land (A_2 \lor A_3)\]
If we assume the failure probability of each leaf node is 0.5

\[(A_1 \lor A_2) \land (A_2 \lor A_3)\]  

Failure probability = model counter output \/(2^{\text{the number of leaf nodes}})
Model Counter

If we assume the failure probability of each leaf node is 0.5

\[(A_1 \lor A_2) \land (A_2 \lor A_3)\]

Failure probability = \[\frac{5}{(2^3)} = \frac{5}{8}\]
Model Counter

\[(A_1 \lor A_2) \land (A_2 \lor A_3)\]

The probability of Leaf nodes is not 0.5 in practice.

Failure probability = \(\frac{5}{(2^3)} = \frac{5}{8}\)
Model Counter

- Redundant deployment fails
  - S1 fails
    - Agg1 fails
      - 1/2
  - S2 fails
    - Core1 fails
      - 1/8
    - Agg2 fails
      - 1/2
Model Counter

Redundant deployment fails

- S1 fails
  - Agg1 fails
  - Core1 fails
    - Core1a fails
  - Agg2 fails
    - Core1b fails
    - Core1c fails

- S2 fails
  - Agg1 fails
  - Core1 fails
    - Core1a fails
  - Agg2 fails
    - Core1b fails
    - Core1c fails

1/2
1/8
1/2
1/2
1/2
1/2
Model Counter

Redundant deployment fails

- S1 fails
  - Agg1 fails
    - 1/2
  - Core1 fails
    - 1/8
  - Agg2 fails
    - 1/2

- S2 fails

Redundant deployment fails

- S1 fails
  - Agg1 fails
    - 1/2
  - Core1 fails
    - 1/2
  - Agg2 fails
    - 1/2

- S2 fails

- Core1a fails
- Core1b fails
- Core1c fails
Redundant deployment fails

S1 fails

S2 fails

Agg1 fails

Core1 fails

Agg2 fails

Model counter output / \((2^5)\)
Model Counter

Redundant deployment fails

\[ \text{S1 fails} \quad \text{S2 fails} \]

\[ \text{Agg1 fails} \quad \text{Core1 fails} \quad \text{Agg2 fails} \]

1/2

1/7

1/2
The algorithm is approximate
Issues in INDaaS

- Hard to express diverse auditing tasks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime
  - Much faster analysis based on SAT solver variants
- Have no idea how to fix the cascading failure problem
  - Automatically generate improvement plans

Diagram:
- Auditing
  - Service initialization
  - Changing network paths
  - Upgrading software components
- Service Runtime
Issues in INDaaS

- Hard to express diverse auditing tasks
  - A new domain-specific auditing language
- Fault graph analysis does not support auditing in runtime
  - Much faster analysis based on SAT solver variants
- Have no idea how to fix the cascading failure problem
  - Automatically generate improvement plans
Repair

Specification:
\[
\text{Server} \rightarrow 172.28.228.21, 172.28.228.22 \\
\text{goal} (\text{failProb}(ft) < 0.08 \mid \text{ChNode} \mid \text{Agg3})
\]
Repair

Specification:

$Server \rightarrow 172.28.228.21, 172.28.228.22$

$\text{goal}(\text{failProb}(ft) < 0.08 \mid \text{ChNode} \mid \text{Agg3})$
$Server -> 172.28.228.21, 172.28.228.22$

goal(failProb(ft)<0.08 | ChNode | Agg3)

**Specifcation:**

**Repair Engine**

Plan 1: Move replica from S1 -> S4
Plan 2: Move replica from S2 -> S4
Repair

Specification:
$Server \rightarrow 172.28.228.21, 172.28.228.22$
$\text{goal}(\text{failProb}(ft)<0.08 \mid \text{ChNode} \mid \text{Agg3})$

Plan 1: Move replica from S1 $\rightarrow$ S4
Plan 2: Move replica from S2 $\rightarrow$ S4
Repair

**Specification:**

\[ \text{goal} \left( \text{failProb}(\text{ft}) < 0.08 \mid \text{ChNode} \mid \text{Agg3} \right) \]

Plan 1: Move replica from S1 → S4

Plan 2: Move replica from S2 → S4
Evaluation

- Realistic case studies.
- Evaluating expressiveness of our language
- Comparing fault graph analysis algorithms
- Evaluating efficiency of repair engine
- . . . . . .
Evaluation

• Realistic case studies.
• Evaluating expressiveness of our language
• Comparing fault graph analysis algorithms
• Evaluating efficiency of repair engine
• . . . . .
## Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Topology A</th>
<th>Topology B</th>
<th>Topology C</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Core Routers</td>
<td>144</td>
<td>576</td>
<td>1,024</td>
</tr>
<tr>
<td># of Agg Switches</td>
<td>288</td>
<td>1,152</td>
<td>2,048</td>
</tr>
<tr>
<td># of ToR Switches</td>
<td>288</td>
<td>1,152</td>
<td>2,048</td>
</tr>
<tr>
<td># of Servers</td>
<td>3,456</td>
<td>27,648</td>
<td>65,536</td>
</tr>
<tr>
<td>Total # of devices</td>
<td>4,176</td>
<td>30,528</td>
<td>70,656</td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Topology A</th>
<th>Topology B</th>
<th>Topology C</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Core Routers</td>
<td>144</td>
<td>576</td>
<td>1,024</td>
</tr>
<tr>
<td># of Agg Switches</td>
<td>288</td>
<td>1,152</td>
<td>2,048</td>
</tr>
<tr>
<td># of ToR Switches</td>
<td>288</td>
<td>1,152</td>
<td>2,048</td>
</tr>
<tr>
<td># of Servers</td>
<td>3,456</td>
<td>27,648</td>
<td>65,536</td>
</tr>
<tr>
<td>Total # of devices</td>
<td>4,176</td>
<td>30,528</td>
<td>70,656</td>
</tr>
</tbody>
</table>
Topology C: 70,656 Nodes

Accuracy

Computational time (minutes)
Topology C: 70,656 Nodes

The top-20 critical RCGs detected

Computational time (minutes)

Accuracy

Minimal Cut Set Algorithm
Topology C: 70,656 Nodes

The top-20 critical RCGs detected

Computational time (minutes)

Accuracy

INDaaS (10^5 rounds)

Minimal Cut Set Algorithm

INDaaS (10^6 rounds)

INDaaS (10^7 rounds)
Topology C: 70,656 Nodes

The top-20 critical RCGs detected

Computational time (minutes)

Accuracy

RepAudit

Minimal Cut Set Algorithm

INDaaS (10^5 rounds)

INDaaS (10^6 rounds)

INDaaS (10^7 rounds)
Our approach is 300x faster than INDaas, and offers 100% accurate results.
• INDaaS is the first system preventing correlated failures
  - Automatically collecting dependency data
  - Reasonable abstraction: Fault graph

• RepAudit is a language framework auditing correlated failures in system runtime:
  - Flexible to express diverse auditing tasks
  - Accurate and rapid auditing capabilities
  - Useful to build new applications (e.g., repair)