Towards An Auditing Language for Preventing Cascading Failures

Ennan Zhai and Ruzica Piskac

Department of Computer Science

Yale
Background

• Cloud services ensure reliability by redundancy:
  - Storing data redundantly
  - Replicating service states across multiple nodes
Background

• Cloud services ensure reliability by redundancy:
  - Storing data redundantly
  - Replicating service states across multiple nodes

• Examples:
  - Microsoft Azure, Amazon AWS, Google, etc. replicate their data and service states
Correlated failures resulting from EBS due to bugs in one EBS server

Summary of the October 22, 2012 AWS Service Event in the US-East Region

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)
Summary of the October 22, 2012 AWS Service Event in the US-East Region

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)

Correlated failures resulting from EBS due to bugs in one EBS server

However, cloud outages still occur!

Correlated failures resulting from EBS due to bugs in one EBS server

Rackspace Outage Nov 12th

On November 12th at 13:51 CST Rackspace experienced an isolated issue in their core network. A small number of their customers were affected, including REW. The outage lasted about 90 minutes. In simple terms, a core network switch died and when the traffic failed over to the secondary switch it also died. Rackspace is investigating the incident to find ways to improve their network and processes to ensure this event is not repeated. REW Sysadmins were immediately notified of the outage by our monitoring tools and were in constant contact with Rackspace during the outage working to resolve as quickly as possible.

REW apologizes for this outage; we promise that we are putting Rackspace’s feet to the fire to ensure maximum uptime for our customers!

Here is the incident report from Rackspace if you want the techy details:
However, cloud outages still occur!

Final Root Cause Analysis and Improvement Areas: Nov 18 Azure Storage Service Interruption

Posted on December 17, 2014

Jason Zander, CVP, Microsoft Azure Team

On November 18, 2014, many of our Microsoft Azure customers experienced a service interruption that impacted Azure Storage and several other services, including Virtual Machines. Following the incident, we posted a blog that outlined a preliminary Root Cause Analysis (RCA), to ensure customers understood how we were working to address the issue. Since that time, our highest priority has been actively investigating and mitigating this incident. Today, we’re sharing our final RCA, which includes a comprehensive outline of steps we’ve taken to mitigate against this situation happening again, as well as steps we’re taking to improve our communications and support response. We sincerely apologize and recognize the significant impact this service interruption may have had on your applications and services. We appreciate the trust our customers place in Microsoft Azure, and I want to personally thank everyone for the feedback which will help our business continually improve.

Root Cause Analysis

On November 18th [PST] (November 19th [UTC]) Microsoft Azure experienced a service interruption that resulted in intermittent connectivity issues with the Azure Storage accounts in multiple locations. Relevant services, primarily

Here is the incident report from Rackspace if you want the techy details:
However, cloud outages still occur!

Why replications do not help?

Final Root Cause Analysis and Impact Story

On November 18th [PST] (November 19th [UTC]) Microsoft Azure experienced a service interruption that resulted in intermittent connectivity issues with the Azure StorSimple tape in multiple accounts. Important services, primarily

Here is the incident report from Rackspace if you want the techy details:
Cascading Failures!


Cascading Failures in Reality

Elastic Compute Cloud (EC2)

Elastic Block Store (EBS)
Cascading Failures in Reality

Elastic Compute Cloud (EC2)

EBS Cluster 1

EBS Cluster 2
Existing Approaches

- Cloud providers handle cascading failures via:
  - Provenance systems (e.g., Y! [SIGCOMM’14] and ExSPAN [SIGMOD’10]);
  - Troubleshooting systems (e.g., Sherlock [SIGCOMM’07]).
Existing Approaches

• Cloud providers handle cascading failures via:
  - Provenance systems (e.g., Y! [SIGCOMM’14] and ExSPAN [SIGMOD’10]);
  - Troubleshooting systems (e.g., Sherlock [SIGCOMM’07]).

• Solving the problem after outage occurs.
Existing Approaches

• Cloud providers handle cascading failures via:
  - Provenance systems (e.g., Y! [SIGCOMM’14] and ExSPAN [SIGMOD’10]);
  - Troubleshooting systems (e.g., Sherlock [SIGCOMM’07]).

• Solving the problem *after* outage occurs.

• Prolonged recovery time in complex systems.
Existing Approaches

- Cloud providers handle cascading failures via:
  - Provenance systems (e.g., Y! [SIGCOMM’14] and ExSPAN [SIGMOD’10]);
  - Troubleshooting systems (e.g., Sherlock [SIGCOMM’07]).

- Solving the problem *after* outage occurs.
- Prolonged recovery time in complex systems.
- Cannot avoid system downtime.
Existing Approaches

- Cloud providers handle cascading failures via:
  - Provenance systems (e.g., Y! [SIGCOMM’14] and ExSPAN [SIGMOD’10]);
  - Troubleshooting systems (e.g., Sherlock [SIGCOMM’07]).
- Solving the problem after outage occurs.
- Prolonged recovery time in complex systems.
- Cannot avoid system downtime.

*Disease prevention is better than diagnosis*  
-- *World Health Organization*
Our goal: Preventing cascading failures!

- INDaaS: First effort towards this goal [OSDI’14]
  
  *Heading off correlated failures through Independence-as-a-Service*

- An auditing language framework [OOPSLA’17]
  
  *An auditing language for preventing correlated failures within the clouds*
Service initialization

Service Runtime

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

INDaaS [OSDI’14]

Service initialization

Post-Failure Forensics

1. Diagnosis tools
2. Accountability
3. Provenance
4. ... ...

Service Runtime
Our Initial Work: INDaaS [OSDI’14]

- INDaaS does pre-deployment recommendations:
Our Initial Work: INDaaS [OSDI’14]

- INDaaS does pre-deployment recommendations:
  - Step1: Automatically collecting dependency data
Our Initial Work: 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
Redundancy configuration fails
Redundancy configuration fails

- Server 1 fails
- Server 2 fails

**AND gate:** all the sublayer nodes fail, the upper layer node fails
Redundancy configuration fails

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

Server 2 fails
- Net fails
- SW fails
- HW fails

OR gate: one of the sublayer nodes fails, the upper layer node fails
Redundancy configuration fails

Server 1 fails
- Hardware fails
- Network fails
- Software fails
  - Path 1
  - Path 2
    - Agg 1
    - Core 1
    - Agg 2

Server 2 fails
- Hardware fails
- Network fails
- Software fails
  - HBase
  - HDFS

...
Redundancy configuration fails

Server 1 fails

- HW fails
- Net fails
- SW fails

Path1

- Agg1
- Core1
- Agg2

1 or 0

Server 2 fails

- Net fails
- SW fails
- HW fails

HBase

- ... ...

HDFS

- ... ...

1 or 0
Redundancy configuration fails

Server 1 fails

- HW fails

- Net fails

Path1

- Agg1

- Core1

- Agg2

SW fails

Server 2 fails

- Net fails

- SW fails

HBase

HDFS

HW fails

Net fails

SW fails

... ...

... ...

... ...

... ...

✔

✔

✘

✘

✘

✘

✘

✘

✔

✔

✘
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 INDaaaS

- 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]

Issues in INDaaS

- Service initialization
- Changing network paths
- Upgrading software components

Service Runtime
Our Solution: 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
Our Solution: 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

Service Runtime

- Service initialization
- Changing network paths
- Upgrading software components
Our Solution: 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

![Diagram showing the flow of service runtime and auditing tasks: Service initialization, changing network paths, upgrading software components, and auditing.]
Identifying Unexpected Dependencies
Identifying Unexpected Dependencies
CNF = (A₂) \( \land \) (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"]}

Core Router1
Core Router2
Agg Switch3
Server2 (S2)
172.28.228.22
Server3 (S3)
172.28.228.23
Agg Switch1
Agg Switch2
Server (S1)
172.28.228.21
HDFS
HBase
Server4 (S4)
172.28.228.24
Core Router1
Core Router2
Agg Switch3
Agg Switch2
Agg Switch1
10.0.0.1
10.0.0.2
10.0.0.3
75.142.33.98
75.142.33.99
Internet

Service Deployment
(network/software stacks)
Auditing Engine
Auditing Program in RAL
Auditing Results
Weighted
MaxSAT solver
Replication
Replica1
Replica2
A
A
A
<Weight Vector>

Service Deployment
(network/software stacks)
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
CNF = (A_2) \land (A_1 \lor A_3)

let Server("172.28.228.21") -> s_1
let Server("172.28.228.22") -> s_2
let [s_1, s_2] -> rep
let FaultGraph(rep) -> ft
let RankRCG(ft, 2, NET, ft) -> ranklist

Auditing Program

Auditing Engine

INDaaS data collection

Replication

Replica1

Replica2

A_1

A_2

A_3

HBase

HDFS

HBase

HDFS

HBase

HDFS

Service Deployment (network/software stacks)
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 ˅ A3)

Auditing Program

Auditing Engine

INDaaS data collection

Replication

Weighted MaxSAT solver

CNF = (A2) ˄ (A1 ˅ A3)

<Weight Vector>
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

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

Auditing Results

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

INDaaS data collection

Replication

Weighted MaxSAT solver

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

<Weight Vector>
RepAudit’s 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"]}

Core Router1
Core Router2
Agg Switch1
Agg Switch2
Agg Switch3
Server2 (S2) 172.28.228.22
Server3 (S3) 172.28.228.23
Server4 (S4) 172.28.228.24
10.0.0.1
10.0.0.2
10.0.0.3
75.142.33.98
75.142.33.99

INDaaS data collection

HBase
HDFS

HBase
HDFS

Replication
Replica1
Replica2
A1
A2
A3

Weighted MaxSAT solver

CNF = (A2) ∧ (A1 v A3)
<Weight Vector>

Service Deployment
(network/software stacks)
RepAudit’s 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”]}

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

CNF = (A2) ∧ (A1 ∨ A3)

<Weight Vector>

Weighted MaxSAT solver

Replication

Replica1

Replica2

indiass

INDaaS data collection

HBase

HDFS

HBase

HDFS

Server1 (S1) 172.28.228.21

Agg Switch1 (Agg1) 10.0.0.1

Core Router1 (Core1) 75.142.33.98

Server2 (S2) 172.28.228.22

Agg Switch2 (Agg2) 10.0.0.2

Server3 (S3) 172.28.228.23

Agg Switch3 (Agg3) 10.0.0.3

Core Router2 (Core2)

Server4 (S4) 172.28.228.24

10.0.0.3

HDFS

HDFS

Service Deployment (network/software stacks)

Internet

Service Deployment (network/software stacks)
Auditing 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.

\[ e ::= g \mid c \mid l(e) \mid q \mid e_1 \text{ op } e_2 \quad \text{Expression} \]
\[ c ::= i \mid \text{str} \quad \text{Real number or string} \]
\[ l(e) ::= \text{nil} \mid [e_1, \ldots, e_n] \quad \text{List} \]
\[ op ::= < \mid \leq \mid \geq \mid != \mid > \mid \geq \quad \text{Operator} \]
\[ q ::= \text{Server}(e) \quad \text{Initializing server node} \]
| \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.
Auditing Language

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

(a) Statements of RAL.

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

(b) Expressions of RAL.
Identifying Shared Dependencies

Auditing Program

```plaintext
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"]}

Weighted MaxSAT solver

CNF = (A2 ∧ (A1 ∨ A3))

<Weight Vector>

Service Deployment (network/software stacks)
Risk Groups in Fault Graphs
Risk Groups in Fault Graphs

A risk group means a set of leaf nodes whose simultaneous failures lead to the failure of root node.
Risk Groups in Fault Graphs

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
State-of-the-Art Risk Group Analysis

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

- State-of-the-art risk group detection efforts:
  - Deterministic minimal cut set algorithm
    - Pros: Efficient auditing approach
    - Cons: Accuracy is quite low in large system auditing
  - Failure sampling algorithm
    - Pros: 100% accurate results
    - Cons: Exponential-time complexity

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

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., \{A_1=0, A_2=1, A_3=0\} represents a risk group.
Our Insight

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

\{A_1=0, A_2=1, A_3=0\}

SAT solver
Our Insight

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

<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>0</td>
<td></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>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Discovering 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

### Table Example

<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>1</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</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>3</td>
</tr>
</tbody>
</table>
Discovering 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

<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>1</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>
Discovering Risk Groups

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


Discovering Risk Groups

- Find out the top-k critical risk groups
  - Use a $\wedge$ 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>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.03</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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></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></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></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.006</td>
</tr>
</tbody>
</table>
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](image)
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 showing service runtime with nodes for service initialization, changing network paths, and upgrading software components connected to auditing]
Repair
$Server \rightarrow 172.28.228.21, 172.28.228.22$

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

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

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

Repair Engine

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

Specification:
$Server \rightarrow 172.28.228.21, 172.28.228.22$

\(\text{goal(failProb(ft)<0.08 | ChNode | Agg3)}\)

**Plan 1:** Move replica from S1 \(\rightarrow\) S4

**Plan 2:** Move replica from S2 \(\rightarrow\) S4
$Server -> 172.28.228.21, 172.28.228.22$

**goal**($\text{failProb}(ft)<0.08 \mid \text{ChNode} \mid \text{Agg3})$

---

**Plan 1:** Move replica from S1 -> S4

**Plan 2:** Move replica from S2 -> S4

---

**Synthesis**

**Repair Engine**
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
• . . . . .
# Fault Graph Analysis

<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>
## Fault Graph Analysis

<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>
The top-20 critical RCGs detected

Accuracy

Computational time (minutes)

Minimal Cut Set Algorithm

Topology C: 70,656 Nodes
Topology C: 70,656 Nodes

The top-20 critical RCGs detected

Computational time (minutes)

Accuracy

Minimal Cut Set Algorithm

INDaaS (10^5 rounds)

INDaaS (10^6 rounds)

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

- 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.
Conclusion

- INDaaS is the first system preventing cascading failures
  - Automatically collecting dependency data
  - Reasonable abstraction: Fault graph

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

- INDaaS + RepAudit: Preventing cascading failures

- Find source code at:
  - [http://github.com/ennanzhai/repaudit](http://github.com/ennanzhai/repaudit)

- I will be at the poster session
Cascading Failure Example
Cascading Failure Example
Cascading Failure Example
Cascading Failure Example

- **Rack 1**: Switch 1, Replica
- **Rack 2**: Switch 2, Replica
- **Rack 3**: Switch 3, Replica

The diagram illustrates the cascading failure example, where a failure in the aggregation switch (Agg Switch) results in a failure across multiple racks due to the interdependencies between the racks and switches.
Cascading Failure Example

Rack 1
- Replica

Rack 2
- Replica

Rack 3
- Replica

Switch 1

Switch 2

Switch 3

Agg Switch
Cascading Failure

- Rack 1
  - Replica
  - Switch 1
- Rack 2
  - Replica
  - Switch 2
- Rack 3
  - Replica
  - Switch 3

Agg Switch
Cascading Failures in Reality
Cascading Failures in Reality

• Cascading failures by network root causes
• Cascading failures by software root causes
Cascading Failures in Reality

- Cascading failures by network root causes
- Cascading failures by software root causes
Cascading Failures by Network Errors

Rackspace Outage Nov 12th

2 years ago 1,120 Views

On November 12th at 13:51 CST Rackspace experienced an isolated issue in their core network. A small number of their customers were affected, including REW. The outage lasted about 90 minutes. In simple terms, a core network switch died and when the traffic failed over to the secondary switch it also died. Rackspace is investigating the incident to find ways to improve their network and processes to ensure this event is not repeated. REW Sysadmins were immediately notified of the outage by our monitoring tools and were in constant contact with Rackspace during the outage working to resolve as quickly as possible.

REW apologizes for this outage; we promise that we are putting Rackspace's feet to the fire to ensure maximum uptime for our customers!

Here is the incident report from Rackspace if you want the techy details:

Cascading Failures in Reality

- Cascading failures by network root causes
- Cascading failures by software root causes
Discovering Critical Risk Groups

- Discovering the top-k risk groups with the highest failure probabilities
  - We want to maximize $C = \prod c_i \cdot w_i$ rather than $C = \sum c_i \cdot w_i$
  - Use $(-100)\log c_i$ as the cost

\[
\{A1=0.2, A2=0.6\} \text{ and } \{A3=0.39, A3=0.39\}
\]

\[
\log 0.2 + \log 0.6 = \log 0.2 \times 0.6
\]

\[
\log 0.39 + \log 0.39 = \log 0.39 \times 0.39
\]
Discovering Critical Risk Groups

- Discovering the top-k risk groups with the highest failure probabilities
  - We want to maximize $C = \prod c_i \cdot w_i$ rather than $C = \sum c_i \cdot w_i$
  - Use $(-100)\log c_i$ as the cost

Our engine is 300x faster than INDaaS, and offers 100% accurate results.

\[
\log 0.2 + \log 0.6 = \log 0.2 \cdot 0.6
\]

\[
\log 0.39 + \log 0.39 = \log 0.39 \cdot 0.39
\]
First-Step System: INDaaS [OSDI’14]