

Towards An Auditing Language for Preventing Cascading Failures

Ennan Zhai and Ruzica Piskac
Yale University

“Disease prevention is better than diagnosis.” – World Health Organization [2]

1. BACKGROUND & MOTIVATION

Today’s cloud services heavily rely on replication techniques to ensure reliability. As the underlying structures of cloud services become complex, infrastructure components may unwittingly share deep dependencies. These unexpected common dependencies’ faults may result in *cascading failures* across the entire replication deployments, undermining the reliability enhancement efforts [15, 9, 10, 16].

In a recent Rackspace outage event [3], for example, the glitches on a core switch and its backup switch in the Rackspace cloud caused multiple servers to be inaccessible, thus blocking customers from accessing their data in Rackspace. A survey conducted by Gunawi *et al.* [10] based on 500+ real-world cloud outage reports revealed that cascading failures resulting from only a small set of network-component faults (called *network weak spots*) are the second most common type of cloud outages.

Moreover, Dan Geer, a computer risk management specialist, pointed out *“A more insidious source of common-mode failures is a design fault, e.g., a killer bug, that causes redundant copies of the same software process to fail under identical conditions.”* [8]. For example, in an Amazon AWS report [1], a killer bug on one Amazon EBS data collector caused cascading failures across Relational Database Storage (RDS) and its backups, affecting the entire Availability Zone.

There have been many diagnostic and forensic efforts proposed to localize the root causes of network problems [6, 11, 13] or software problems [5] after service outages occur. However, these post-failure forensics require significant human interventions, and in the face of structurally complex services, this leads to excessively prolonged failure recovery time [14]. Therefore, a natural question is: rather than trying to detect the root cause of a failure after it happens, can we prevent cascading failures before they occur?

2. RepAudit FRAMEWORK

We introduce RepAudit, a novel framework that aims at preventing cascading failures before cloud outages occur. The main idea is to allow administrators to proactively audit

the underlying structures of a given replication deployment. RepAudit enables an administrator to easily express heterogeneous auditing tasks (*e.g.*, identifying network weak spots and killer bugs potentially causing cascading failures). In addition, RepAudit can also automatically generate new deployment plans for improving the reliability of the original replication deployment. With RepAudit, administrators can understand potential cascading failure risks at an early stage.

We identify the following main technical challenges in building RepAudit.

1. Auditing language. Currently administrators either write risk-analysis scripts manually, which is error-prone and tedious, or adapt existing analysis tools to their specific purposes. In both cases, it is hard to support heterogeneous auditing tasks, since administrators need to fully understand complex underlying structures of audited systems. To address this issue, we propose a declarative domain-specific auditing language, RAL. It is a high-level language that hides the details of system structures, so that administrators can easily express their auditing tasks. An example of an RAL-program is given in the top-left corner in Figure 1. RAL programs are parsed and executed by the auditing engine. Inside the auditing engine, as shown in Figure 1, any target system’s underlying structures are modeled using a fault graph [15]. Broadly speaking, it is a data structure representing a system as a directed acyclic graph (DAG) with logical gates.

2. Efficient and accurate auditing engine. RepAudit fundamentally differs from post-failure efforts that try to localize root causes specific to observed outages. Proactive auditing is required to be both accurate and efficient, because it needs to search and assess cascading failure root causes in the entire underlying structures of audited systems. For the structurally complex cloud systems with tens of thousands of components and multi-layered hardware/software stacks, it is challenging to make fault graph analysis approaches achieve both accuracy and efficiency. Inspired by successful applications of SAT/SMT solvers to formal verification of large-scale programs, we leverage them to construct fault graph analysis algorithms. We mainly use weighted partial MaxSAT solver [4], model counter [7] and network transformation [12]. These analysis algorithms are bases for the language primitives in the auditing engine. Because of using

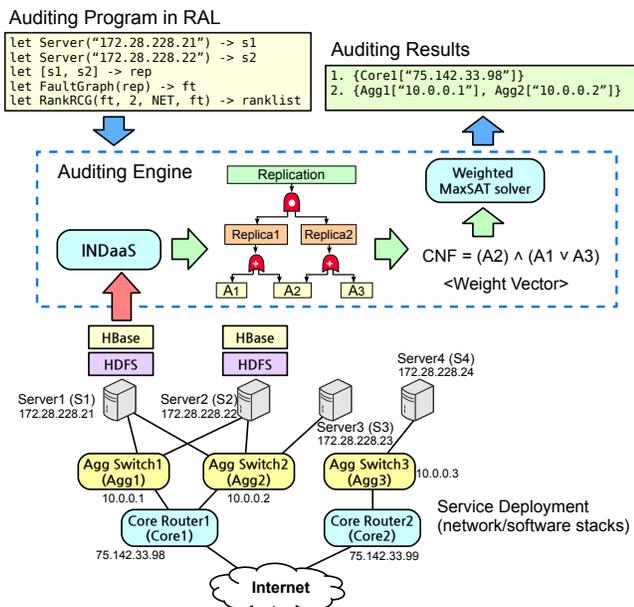


Figure 1: A RepAudit workflow example.

these formal reasoning tools, RepAudit can simultaneously offer accurate and efficient structural reliability auditing.

Figure 1 shows an example (similar to the Rackspace situation in §1). An administrator wants to proactively identify network weak spots (specified by NET in RankRCG()) that could potentially result in cascading failures. The administrator first expresses her auditing task by submitting the auditing engine an auditing program in RAL. Then, the auditing engine uses INDaaS [15] to automatically generate a fault graph modeling the systems of interest, and transforms the generated fault graph into a Boolean formula in Conjunctive Normal Form (CNF). Finally, the engine calls a weighted partial MaxSAT solver to create the auditing results. They are presented to the administrator and she understands that the core router (*i.e.*, Core1) is the “bottleneck” in her service.

In our preliminary evaluation, we run RepAudit on a real-world cloud storage dataset consisting of 70,656-nodes. We observe that RepAudit determines the top-20 critical root causes 300× more efficient than the state-of-the-art auditing tool [15]. Furthermore, the returned results of RepAudit are guaranteed to be 100% accurate within the given fault graph.

3. Automatically improving risky deployments. If an administrator notices that their replication deployments have cascading failure risks, trying to manually improve the deployments is again an error-prone process. We, therefore, extend RAL so that administrators can easily specify their improvement goals. We also equip RepAudit with a repair engine, which can automatically generate improvement plans corresponding to those specified goals.

As an illustration, consider the example given in Figure 1. If the administrator wants to improve her current deployment to have failure probability lower than 0.05, she can

convey to the repair engine the following specification: goal (failProb(ft) < 0.05 | ChNode | Agg3). This specification states that the repair engine should generate plans for making the failure probability of the replication deployment lower than 0.05. Additionally, the specification also states that we require moving a replicated state to more independent replica servers (thus, the ChNode option), and the new deployment must contain the switch Agg3. Taking this specification as input, the repair engine generates two alternative plans.

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

Server S4 is structurally independent from other servers (see Figure 1). The repair engine outputs plans that are guaranteed to result in a topology that has the failure probability lower than 0.05.

3. SUMMARY

In this paper we have presented the basic ideas behind a framework RepAudit for preventing cascading failures. RepAudit helps administrators to identify potential root causes resulting in cascading failures at an early stage, *i.e.*, before service outages occur. Due to the application of the state-of-the-art SAT/SMT solvers, RepAudit is capable of achieving both efficient and accurate structural reliability auditing to cloud-scale systems.

4. REFERENCES

- [1] Correlated Failures within EBS and EC2. <https://goo.gl/tFZ9aD>.
- [2] Prevention is better than cure. <https://goo.gl/jtpzse>.
- [3] Rackspace Outage Nov 12th. <https://goo.gl/on6Kb5>.
- [4] Mario Alviano, Carmine Dodaro, and Francesco Ricca. A MaxSAT algorithm using cardinality constraints of bounded size. In *24th IJCAI*, July 2015.
- [5] Mona Attariyan and Jason Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *9th OSDI*, October 2010.
- [6] Paramvir Bahl, Ranveer Chandra, Albert G. Greenberg, Srikanth Kandula, David A. Maltz, and Ming Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *SIGCOMM*, 2007.
- [7] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In *19th CP*, 2013.
- [8] Dan Greer. Heartbleed as metaphor. *Lawfare*, April 2014. <https://goo.gl/nno0XV>.
- [9] Haryadi S. Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tirat Patana-anake, Thanh Do, Jeffry Adityama, Kurnia J. Eliazar, Agung Laksono, Jeffrey F. Lukman, Vincentius Martin, and Anang D. Satria. What bugs live in the cloud? A study of 3000+ issues in cloud systems. In *5th SoCC*, 2014.
- [10] Haryadi S. Gunawi, Mingzhe Hao, Riza O. Suminto, Agung Laksono, Anang D. Satria, Jeffry Adityatama, and Kurnia J. Eliazar. Why does the cloud stop computing? Lessons from hundreds of service outages. In *7th SoCC*, 2016.
- [11] Joshua B. Leners, Hao Wu, Wei-Lun Hung, Marcos Kawazoe Aguilera, and Michael Walfish. Detecting failures in distributed systems with the Falcon spy network. In *23rd SOSP*, 2011.
- [12] Gordon D. Plotkin, Nikolaj Björner, Nuno P. Lopes, Andrey Rybalchenko, and George Varghese. Scaling network verification using symmetry and surgery. In *43rd POPL*, January 2016.
- [13] Patrick Reynolds, Charles Edwin Killian, Janet L. Wiener, Jeffrey C. Mogul, Mehul A. Shah, and Amin Vahdat. Pip: Detecting the unexpected in distributed systems. In *3rd NSDI*, 2006.
- [14] Xin Wu, Daniel Turner, Chao-Chih Chen, David A. Maltz, Xiaowei Yang, Lihua Yuan, and Ming Zhang. NetPilot: Automating datacenter network failure mitigation. In *SIGCOMM*, 2012.
- [15] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. Heading off correlated failures through Independence-as-a-service. In *11th OSDI*, October 2014.
- [16] Ennan Zhai, David Isaac Wolinsky, Hongda Xiao, Hongqiang Liu, Xueyuan Su, and Bryan Ford. Auditing the Structural Reliability of the Clouds. Technical Report YALEU/DCS/TR-1479, Department of Computer Science, Yale University, 2013. Available at <http://cpsc.yale.edu/sites/default/files/files/tr1479.pdf>.