Fault Prediction in Distributed Systems Gone Wild

Marco Canini, Dejan Novaković, Vojin Jovanović, and Dejan Kostić
Networked Systems Laboratory
School of Computer and Communication Sciences, EPFL, Switzerland
firstname.lastname@epfl.ch

ABSTRACT

We consider the problem of predicting faults in deployed, large-scale distributed systems that are heterogeneous and federated. Motivated by the importance of ensuring reliability of the services these systems provide, we argue that the key step in making these systems reliable is the need to automatically predict faults. For example, doing so is vital for avoiding Internet-wide outages that occur due to programming errors or misconfigurations.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; H.4.3 [**Information Systems Applications**]: Communications Applications

General Terms

Reliability

Keywords

Fault prediction, Federated systems, Heterogeneous systems, Shadow snapshot, Spatial and temporal awareness, BGP

1. INTRODUCTION

Large-scale distributed systems are already at the foundation of today's Internet services and continue to grow in popularity and importance. However, making large-scale distributed systems reliable is a notorious challenge. Moreover, many successful systems become heterogeneous due to the creation of multiple implementations and evolve into multi-provider distributed systems as a result of deployment in the wide-area network with several federated administrative domains.

Recent research efforts have focused on finding bugs in distributed systems by applying model checking [15, 22, 23] or symbolic execution [20] to explore a large number of potential states. We argue that making heterogeneous, federated distributed systems reliable is even more challenging because (i) the source code of every node may not be readily available for testing and (ii) competitive concerns are likely to induce individual providers to keep private much of their current state and configuration.

Examples of systems of such nature encompass inter-domain routing, electronic mail, peer-to-peer content distribution [11], content and resource peering [5, 12]. We believe that the rapid evolution of cloud computing will further foster the emergence of new such systems[1].

Motivated by the importance of ensuring dependability of long-running systems that are federated and heterogeneous, we argue for the need to predict faults and assess their impact as this is the crucial step in being able to guard against important classes of faults. A key insight in doing so is that nodes need to become spatially and temporally aware of the consequences of local actions on their neighborhood. We propose to achieve spatial awareness by creating a consistent, shadow snapshot, i.e., a distributed snapshot of the system taken from the current live state in which nodes are allowed to communicate with each other in isolation from the running environment, while respecting the node trust boundaries. Then, we achieve temporal awareness by subjecting the system snapshot to a large number of possible scenarios created by systematically exploring the potential behavior of a node and judging the wider impact of its actions. Finally, we predict the aggregate future behavior across multiple nodes by checking the status of safety properties and system invariants in the shadow snapshots. This approach is illustrated in Figure 1.

In this context, by fault prediction we mean to detect possible sequences of actions which reach states that present inconsistencies that can lead to failures. As these might actually never happen, our prediction is loose with respect to time in that it is not associated to a time window during which the failures could occur. For a thorough discussion on failure prediction we refer the reader to [19]. According to literature [6], faults are the adjudged or hypothesized cause of an error. A failure refers to misbehavior that can be observed by the user. Given that we are exploring possible actions, these include faults. Therefore, because we want to avoid failures, i.e., prevent faults that have a visible erroneous state, we refer to our approach as fault prediction.

Based on this approach, we are building a prototype which is already successful at predicting certain operator mistakes in BGP router configurations.

With this paper we solicit discussions around the key insights of making distributed systems reliable.

The rest of the paper is organized as follows. In Section 2 we motivate the importance of predicting faults. Section 3 presents a num-

[1]For instance, Google has opened to third parties the Google Wave Federation Protocol promoting the creation of interoperable wave providers. `http://www.waveprotocol.org`

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
LADIS '10 Zürich, Switzerland
Copyright 2010 ACM 978-1-4503-0406-1 ...$10.00.

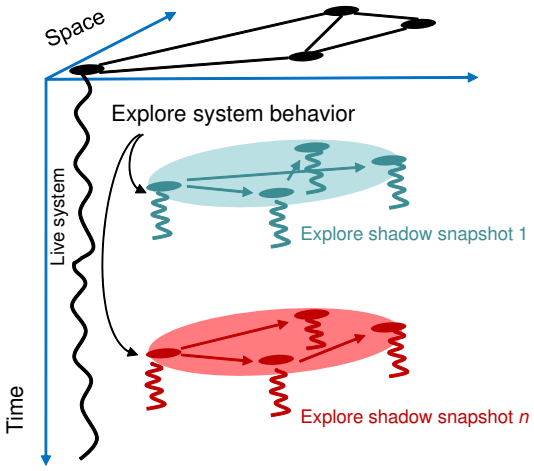

Figure 1: Predicting faults by achieving spatial and temporal awareness. Node behavior is systematically explored and checked over isolated shadow snapshots.

ber of challenges in predicting faults in federated, heterogeneous distributed system. In Section 4 we elaborate on our approach to achieve fault prediction. Finally, Section 5 discusses related work and Section 6 offers concluding remarks and sets the goal for future work.

2. MOTIVATION

The nature of distributed systems, in which the aggregate behavior is the result of interleaved actions of multiple nodes, makes it impossible to check and debug their code in isolation as well as to fully comprehend the overall impact of local actions. Considering the example of inter-domain routing, a BGP session reset in response to a syntactically valid, but semantically useless route announcement might be a perfectly good way of handling of faults for a single router. However, when it is coupled with a large number of routers that propagate the potential fault (because they do not process the update in sufficient detail), the overall effect is a large fraction of the routers that are continuously resetting and restoring sessions. The resulting high update processing rate that is reminiscent of emergent behavior [18] causes a performance and reliability problem [4].

Further, the unanticipated interaction of nodes under seemingly valid configuration changes can have a profound effect. For example, an ISP could use routing policies that lead to divergent routing or have detrimental effect. This was witnessed in a recent episode when Pakistan Telecom mistakenly managed to hijack the vast majority of traffic directed toward YouTube for more than an hour [3]. But because at the core of ISPs profitability is the common business practice of keeping most of the inter-domain policies and intra-domain information private, it would not be possible to convince them to verify suitably defined global invariants even though ubiquitous Internet connectivity is at stake.

Lastly, the fact that a majority of the deployed routers comes from only a handful of vendors [2] makes a large fraction of the Internet susceptible to a single programming error. A malicious packet could be engineered to take down a large fraction of the Internet [1].

Given the importance of the services that distributed systems provide, we believe that the overarching goal of keeping long-running federated distributed systems functioning properly is a significant incentive for the involved providers to become mutually interested in predicting potential faults. However, this entails a number of challenges that we discuss in the next section.

3. CHALLENGES

Heterogeneity.

Distributed systems that are widely deployed are often successful because of the open standards and well-defined interfaces that permit multiple implementations. Further, even the software deployed by the same vendor has multiple versions and patch revisions due to the difficulty of instantaneously upgrading all nodes. Moreover, heterogeneity arises from the lack of global coordination in systems operated under multiple administrative domains. The resulting heterogeneity creates a problem for fault prediction because it is difficult or impossible to have the source (or even binary) code for all nodes that is required to achieve spatial and temporal awareness in existing approaches [22]. Thus, the approach should be able to operate using only the existing interfaces.

Hidden internal state/configuration.

Despite existing business relationships among providers motivates both sides to predict faults, the federated nature of large-scale distributed systems translates to the desire of nodes to keep a large portion of their state and configuration that captures business practices private. This hinders the ability of external entities to identify problems with a node's configuration or software. A solution in this space should only use the well-defined interface and ideally leak no confidential information. It is only in this case that the distributed system operators would have the incentive to participate in the protocol and increase overall system reliability.

Incremental deployment.

While attempting to redesign the protocols and the programming interfaces, a solution that has any chance of success has to be incrementally deployable. This means that the approach should not require changes to the existing protocol messages. Also, it should not pose unnecessary requirements on the programming interfaces or require intrusive changes to the existing systems.

Short running time.

The approach should be able to explore distributed system state during the short quiescent periods. In addition, it should be able to predict potential faults in a timeframe that is short enough for the operators to take preventive measures or the system to automatically adapt. However, a definite trade-off exists between exploration speed and bandwidth consumed. Exploring systems that have a large amount of traffic will require managing the bandwidth used for exploration so that the system performance is not degraded to undesired levels.

Coverage.

Ideally, all code paths or possible states should be explored. This task is difficult even for strictly single-machine software due to the exponential number of paths or states. A typical cause of the excessive number of potential paths is dealing with large inputs [9] (e.g., configuration files). Given the necessity to quickly predict faults in a running system, it becomes important to significantly limit the space that is being searched so that the running time is not prohibitively long. Ideally, the approach should be able to automatically infer message handlers that perform key state transformation and should make use of ad-hoc heuristics which can bias state space

exploration toward states that are more likely to expose faults.

Long running times/large inputs.

The deployed systems we are interested in will potentially be running for months without restarting. This means that the inputs to the system will be long, which further makes achieving path exploration and good coverage difficult.

Scalability.

While the approach would ideally predict system behavior as a whole, doing so has scalability problems of its own. Thus, it is important to devise techniques that work with a constrained number of nodes.

False positives/negatives.

As with any approach that performs fault detection and prediction, false positives and negatives represent a potential issue. False negatives can occur if the properties that are checked for are not capable of discerning the faulty state. Regarding the false positives, live execution over the shadow snapshot is evidence of the behavior that is the result of processing a particular input. However, it is challenging to ensure that the properties themselves are defined in a way that avoids false positives. Also, it is not possible to detect faults that are not checked for. However, automatically inferring system invariants is a substantial challenge per-se [21].

Finally, we note that extrapolating from local to global effects is notoriously difficult and a potential source for false positives. On the other hand, one can leverage specific designs of distributed systems to observe and generalize about faults observed in small settings. For example, DHT nodes typically maintain $\log(N)$ neighbors and problems manifest at that level; whereas in BGP, because global connectivity necessitates that route updates reach every router, faults observed locally will propagate at the global level, likely having effects similar to the local ones.

Snapshotting.

Taking snapshots of a large-scale system can be challenging. To mitigate the resource requirements, fault prediction would make use of resources that are underutilized when the system is not under high load. However, because the snapshot is distributed due to privacy requirements (i.e., each node creates its own checkpoint), the cost of the snapshot can scale with the number of participants. Further, it is important to control how much time it takes to create a snapshot. Two main factors have an influence on it: (i) the propagation delay of the messages that causes each node to take a checkpoint, and (ii) the technique used by each node to actually take the checkpoint.

Also, as a consistent snapshot is desired, the time to generate the snapshot is affected by the message passing strategy of the specific distributed system. An additional challenge is that the frequency at which new snapshots are taken needs to be adapted to how frequently the important pieces of the whole system state change. Sacrificing exploring at sufficient depth the past snapshot so that the most current snapshot can be explored would potentially increase the number of false positives. A key question is what are the faults that can be predicted with the current snapshot that could not be predicted with the previous snapshots. If these are numerous and likely to occur, snapshotting ought to be sufficiently frequent. Finally, as for what state needs to be preserved, our approach requires access to all the state required to run the programs symbolically. We find the system call `fork()` a good technique to make a cheap copy of the process memory.

4. PREDICTING FAULTS

Types of faults.

Among the type of faults that can be detected in distributed systems we target three specific categories: (i) insidious programming errors, (ii) misconfigurations (local), and (iii) conflicting objectives of individual nodes (system-wide misconfiguration).

Spatial awareness.

Because of the heterogeneous and confidential nature of the system states we adopt an approach that respects the node trust boundaries. We let the nodes become spatially aware by establishing a shadow snapshot taken from the current live state. The snapshot itself is distributed and each node is allowed to exchange messages with its neighbors through shadow connections.

Temporal awareness.

In order to predict faults, nodes need to achieve temporal awareness. The key idea is to subject a node to all possible inputs in a way that locally exercises all execution paths and, through shadow messaging with its neighbors, leads to potential, valid system states. The exploration runs online, alongside the deployed system, off the critical execution path. Such exploration draws inspiration from the principle of symbolic execution which is able to systematically explore all possible code paths in a program [9]. Similarly, the proposed approach relies on collecting the constraints that describe which input values can lead to a particular point in the code. Then, it queries a satisfiability solver to determine the values which can take to different paths and executes with those inputs. Each execution continues to collect constraints based on the newly explored paths which enter the composite set of constraints. However, this faces the problem of exponential explosion in the number of possible paths. In addition, having a long running system means that the inputs it has accumulated over time are large. Therefore, we envision that the exploring node would start the exploration from its current, live state to quickly reach possible faulty states.

Checking properties.

Using a set of properties that capture safety among a set of nodes would enable finding faults during exploration. In such a simple scheme, each property can be formulated as a predicate on the state of a single node, which effectively inhibits the possibility of leaking private information. However, sometimes system behavior can not be checked against individual node's states. To allow predicting faults in this conditions, we would want our approach to support safety invariants that check on state spanning multiple nodes. At the same time this contrasts with our desire of avoiding leakage of confidential information.

We propose to cast these invariants into protocols which can be solved using SMPC (Secure Multi-Party Computation) which provably does not leak private information [16]. Although it is challenging to build SMPC solutions that are practical in terms of computation and communication cost, recent work has optimized several operations which made SMPC suitable for processing high volume data in near real-time [7] which we find promising for our work.

5. RELATED WORK

The two fundamental techniques that can be used to predict future system behavior by operating on the source code are model checking and symbolic execution.

Model checking involves exploring system states by executing enabled actions (e.g., timers, message handlers, local actions) in

each encountered state. For distributed systems, the aggregate state is composed of the states of participating nodes. Model checkers also require a harness that can introduce faults, e.g., broken connections, node failures, reordered messages, etc. MaceMC [15] runs state space exploration locally, on a set of state machines initialized from initial state, while CrystalBall [22] instantiates the state machines from live, consistent node checkpoints. MaceMC can determine safety and liveness violations spanning multiple nodes and it was used to find bugs in systems implemented in the Mace [14] distributed systems framework. CrystalBall [22] goes one step further in that it can proactively predict inconsistencies that can occur in a (Mace-based) running distributed system due to unknown programming errors, and effectively prevent them. MODIST [23] goes one step further than MaceMC in that it is capable of model checking unmodified distributed systems. One could potentially use MODIST to orchestrate state space exploration across a cluster of machines in an isolated (non-deployed) scenario. A general issue with model checking techniques is the exponentially large set of potential system states.

Symbolic [9, 10, 13], and the related concolic [8, 17], execution are techniques for systematically exploring the code paths of a given system. The approaches in this space [9, 8] use a constraint solver to force code executions to go down a particular branch. As such, these approaches typically do not require a testing harness when finding bugs in single machine code. Path explosion is the key problem faced by symbolic and concolic execution engines.

To facilitate bug finding in sensor-net code, KleeNet [20] builds a test harness that accommodates messaging and fault injection on top of the Klee symbolic engine [9]. To search for bugs, KleeNet arranges for state-space exploration among the set of TinyOs nodes running in isolation on one machine, prior to deployment.

Our work goes beyond the state-of-the-art in that it predicts faults live, in the challenging conditions encountered during widespread system deployment, e.g., the federated and heterogeneous nature of these environments.

6. FUTURE WORK

It is notoriously difficult to make distributed systems reliable and a key step in doing so is the ability to automatically predict faults. In this paper, we argued that this task becomes even harder in the case of the widely-deployed systems that end up being heterogeneous and federated. We described a lengthy but likely incomplete set of entailed challenges and proposed what we regard as the keystone principles in tackling this problem: achieving spatial and temporal awareness at the nodes in order to predict future aggregate system behavior, and in turn, faults.

Our future research goals are geared toward creating a framework that encompasses all the merits of the works closely related to ours [15, 22, 23, 20] and additionally deals with several important aspects in predicting faults for long-running, heterogeneous and federated distributed systems, including: 1) uncovering faults due to inputs that are different than those fed by the model checking harness, 2) executing without requiring to create and incorporated a fault model into the state-space exploration algorithm, 3) exploring system behavior starting from live state, which is crucial for long-running systems, 4) avoiding leaking confidential information in the messages that are being transmitted, and 5) incorporating the intrinsic heterogeneity of the system. We are building a prototype which is already successful at predicting certain operator mistakes in BGP router configurations.

Acknowledgments

Marco Canini was funded in part by a grant from the Hasler foundation (grant 2103).

7. REFERENCES

[1] How To Build A Cybernuke. `http://www.renesys.com/blog/2010/04/how-to-build-a-cybernuke.shtml`.

[2] Juniper seen regaining core router share in 2008. `http://www.reuters.com/article/idUSN1333533920070613`.

[3] Pakistan hijacks YouTube. `http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml`.

[4] Staring Into The Gorge: Router Exploits. `http://www.renesys.com/blog/2009/08/staring-into-the-gorge.shtml`.

[5] Lisa Amini, Anees Shaikh, and Henning Schulzrinne. Effective Peering for Multi-provider Content Delivery Services. In *Proceedings of the twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '04)*, pages 850–861, Hong Kong, March 2004.

[6] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, January 2004.

[7] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *Proceedings of the 19th USENIX Security Symposium*, Washington, DC, August 2010.

[8] Jacob Burnim and Koushik Sen. Heuristics for Scalable Dynamic Test Generation. Technical Report UCB/EECS-2008-123, EECS Department, University of California, Berkeley, Sep 2008.

[9] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*, pages 209–224, San Diego, CA, December 2008.

[10] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. EXE: Automatically Generating Inputs of Death. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*, Alexandria, Virginia, USA, 2006.

[11] Bram Cohen. Incentives Build Robustness in BitTorrent. In *Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[12] Yun Fu, Jeffrey Chase, Brent Chun, Stephen Schwab, and Amin Vahdat. SHARP: An Architecture for Secure Resource Peering. In *Proceedings of the 19th ACM Symposium on Operating System Principles (SOSP '03)*, Bolton Landing, NY, October 2003.

[13] Patrice Godefroid, Nils Klarlund, and Koushik Sen. DART: Directed Automated Random Testing. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming*

Language Design and Implementation (PLDI '05), New York, NY, USA, 2005.

[14] Charles Killian, James W. Anderson, Ryan Braud, Ranjit Jhala, and Amin Vahdat. Mace: Language Support for Building Distributed Systems. In *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '07)*, San Diego, CA, June 2007.

[15] Charles E. Killian, James W. Anderson, Ranjit Jhala, and Amin Vahdat. Life, Death, and the Critical Transition: Finding Liveness Bugs in Systems Code. In *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI '07)*, Cambridge, MA, April 2007.

[16] Sridhar Machiraju and Randy H. Katz. Verifying Global Invariants in MultiProvider Distributed Systems. In *Proceedings of the Third Workshop on Hot Topics in Networks (HotNets-III)*, San Diego, CA, November 2004.

[17] Rupak Majumdar and Koushik Sen. Hybrid Concolic Testing. In *Proceedings of the 29th International Conference on Software Engineering (ICSE '07)*, pages 416–426, Minneapolis, MN, 2007.

[18] Jeffrey C. Mogul. Emergent (Mis)behavior vs. Complex Software Systems. In *Proceedings of the 2006 EuroSys conference*, Leuven, Belgium, 2006.

[19] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Computing Surveys*, 42(3):1–42, March 2010.

[20] Raimondas Sasnauskas, Olaf Landsiedel, Muhammad Hamad Alizai, Carsten Weise, Stefan Kowalewski, and Klaus Wehrle. KleeNet: Discovering Insidious Interaction Bugs in Wireless Sensor Networks Before Deployment. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN '10)*, pages 186–196, Stockholm, Sweden, 2010.

[21] Maysam Yabandeh, Abhishek Anand, Marco Canini, and Dejan Kostic. Almost-Invariants: From Bugs in Distributed Systems to Invariants. Technical Report NSL-REPORT-2009-007, EPFL, 2009.

[22] Maysam Yabandeh, Nikola Knežević, Dejan Kostić, and Viktor Kuncak. Crystalball: Predicting and preventing inconsistencies in deployed distributed systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*, Boston, MA, April 2009.

[23] Junfeng Yang, Tisheng Chen, Ming Wu, Zhilei Xu, Xuezheng Liu, Haoxiang Lin, Mao Yang, Fan Long, Lintao Zhang, and Lidong Zhou. MODIST: Transparent Model Checking of Unmodified Distributed Systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*, Boston, MA, April 2009.