A root cause localization model for large scale systems

Emre Kıcıman
Stanford University
emrek@cs.stanford.edu

Lakshminarayanan Subramanian
University of California, Berkeley
lakme@cs.berkeley.edu

Abstract

Root cause localization*, the process of identifying the source of problems in a system using purely external observations, is a significant challenge in many large-scale systems. In this paper, we propose an abstract model that captures the common issues underlying root cause localization and hence provides the ability to leverage solutions across different systems.*

1. Introduction

Many large-scale systems, as diverse as Internet service clusters, inter-domain routing in the Internet and software systems, suffer from a common problem: when the system fails to function properly, it is often difficult to determine which part of the system is the source of the problem. The challenge is that, often times, the symptoms of a failure manifest as *end-to-end failures* in the operation of the system as a whole, without causing obvious failures in the system's pieces; simply noticing that something has gone wrong is not enough to tell us where to look to fix it. The consequences are serious: poor fault isolation, slow recovery from failures, and generally unreliable systems.

We define *root cause localization* as the process of determining the possible locations of a problem in a system by analyzing externally visible system behavior. Root cause localization answers the question "Where is the problem?", though not necessarily the harder question, "Why?" For any complex distributed system that operates at a high scale, root cause localization is a fundamental necessity given that internal diagnosis of different aspects of the system is often a very laborious task. The communities that build these large-scale systems have historically taken different approaches to solving this problem—alternatively referred to as fault diagnosis, alarm correlation, root cause analysis, and bug isolation in the context of a wide variety of systems [3, 4, 2, 20, 11, 7, 18, 5]. Despite this, we take the position that many of the underlying challenges to root cause localization are common across a surprisingly diverse set of these systems.

In this paper, we define an abstract model of the root cause localization problem, attempting to capture the commonalities of this problem across different systems. Our model explicitly represents the nature of end-to-end failures, captures the common theoretical and computational challenges, and separates system-specific challenges into the process of mapping a system representation into the abstraction.

We believe that root cause localization is just one aspect of a broader issue: developers are increasingly building large and complex systems whose inner workings are poorly understood. This incomprehensibility leads to unpredictable emergent behaviors, unreliability and poor manageability. The challenges posed by complexity are common to many systems and many solutions, including theoretical and machine learning approaches, may be common as well.

Our explicit goal in presenting this model of root cause localization is to build a bridge that enables researchers across different communities to reason about these common solutions. In particular, we hope to enable and attract theory and machine learning researchers to attack the problem. To highlight this promise, we show how one can leverage existing techniques to solve specific aspects of the general problem and briefly illustrate how these solutions have been applied in three distinct application domains.

2. Example applications

In this section, we ground our approach by considering the root cause localization problem in the context of three very different systems: root cause analysis of Internet routing dynamics, failures in clustered Internet services, and the bug isolation problem in software systems.

2.1. Root cause analysis of BGP dynamics

Understanding the dynamics of Internet routing and pinpointing the source of routing problems is critical to address many of the shortcomings of the Border Gateway Protocol (BGP), the *de facto* interdomain routing protocol. Observing a route update is a clear symptom that some event has occurred. A BGP health inferencing system [3, 4, 9] that performs root cause analysis of BGP dynamics uses data collection centers like Routeviews [21] and RIPE [19], which continuously receive streams of route updates from multiple vantage points. Each route update is associated with path-vector information describing the entire path at the granularity of Autonomous systems (AS's). The underlying root cause localization problem in BGP can be stated as follows: *Given route updates observed at multiple vantage points, determine the potential set of locations of events (at the granularity of AS's) that could have triggered each route update.*

2.2. Failures in Internet service clusters

Today's Internet services (e-commerce, search engines, enterprise applications and others) commonly suffer from *brown-outs*, where part of the functionality of a site goes down or is unavailable, resulting in the failure of user requests. It is critical to quickly determine the source of such problems to reduce the overall downtime of the system. While certain types of failures such as a process crash are easy to detect, the challenging aspect arises when the only detectable symptom of a failure is an end-to-end failure, *e.g.,* a front-end web server observes users' failed HTTP requests.

Large Internet services are usually built using clusters of machines (from 100s to over 50000 machines [15]) divided into multiple tiers (a front-end tier of web servers, multiple tiers of application logic, and a back-end tier of persistent storage) and a user's HTTP request usually traverses most of the tiers in the system. To aid in the diagnosis of such a large system, it is becoming common practice to dynamically record and log the *path* of a request (the machines and services used to fulfill the request) [6, 1]. The root cause localization problem in such a system can be formulated as: *Given the paths of both successful and failed requests, determine the set of components most likely to have caused the failures.*

2.3. Bug isolation

Bugs are practically guaranteed to exist in any large software system. Bugs that are deterministic and easily reproduced are relatively easy to track down and fix. Other bugs, humorously named *heisenbugs* [10], are non-deterministic and quite difficult to track down, even with the latest debugging tools. Recently, Liblit *et al.* have proposed an approach they call *statistical debugging* [14], where they advocate constant sampling of the code-level behaviors of end-user software during normal execution. These behaviors include the results of conditional tests, the return values of functions, etc. By discovering which of these behaviors are most correlated with symptoms of a heisenbug, statistical debugging helps programmers understand and discover a bug's true cause.

Statistical debugging is a direct counterpart to the root cause localization problem in other systems, and can be restated as: *Given the code-level behaviors associated with both correct and buggy executions of a program, determine what parts of the code are most likely to have caused the bug.*

3. Root cause localization problem

In this section, we define a basic form of the root cause localization problem and relate it to the example applications described before.

3.1. System model

We abstractly model a large scale system simply as a collection of known components interacting with each other. Components are the basic granularity for localizing failures. While a clear characterization of the set of components is dependent on the system/application under consideration, the definition of a component should ideally satisfy three properties: (a) component granularity should be fine enough to be a useful marker in locating failures, but not so fine that the system model becomes intractably complex; (b) all components when considered together should completely represent the system; (c) a component, as a whole, should be visible to an instrumentation box diagnosing the system.

Given that failures are externally visible only as end-to-end failures, we assume that misbehavior of individual components is not diagnosable in isolation. The only observations that are visible to external instrumentation are what we define as *quarks* - the smallest end-to-end observable unit of a failure or success. Each quark represents a tuple consisting of: (a) the set of components used by the quark; (b) a *health result*, signifying the failure or success of the quark.

We now revisit the three examples and define the components and quarks in them. In BGP, we associate two different types of components: AS's and inter-AS links. This is primarily to distinguish between events triggered across inter-AS boundaries (*e.g.,* peering link failure) and internal routing events within an AS. Every route observed at a vantage point represents a quark. Stable routes represent healthy quarks and any prefix that is updated is an unhealthy quark signifying the occurrence of a routing event.

For Internet service clusters, the machines and software services running on them represent the system components and every HTTP request represents a quark. The service path of an HTTP request (machines and services used by the request) represent the quark's components and HTTP error monitors act as external detectors to determine the health of every quark.

In software programs, a code module that performs a specific functionality is a component and the code-level behavior corresponding to every execution of the program represents a quark. A correct program execution is a successful quark and a buggy execution is an unsuccessful quark. Note that every execution of the code traverses a different set of components depending on the system environment and input parameters.

Our system model makes two basic assumptions: (a) the system is decomposable into components; (b) the components associated with every quark are externally visible. While not all large-scale systems fit this model, our belief is that these assumptions hold for many such systems that exist today. Our model also explicitly abstracts several system-specific concerns which need to be addressed in the process of transforming a physical system model into the component-quark model. First, determining the components and quarks of a system is very specific to the structure of the system and needs to be addressed on a case by case basis. Second, the association between a quark and its set of used components should represent how faults propagate from their source to their symptoms. Without a reasonable approximation of this fault propagation, the failed quarks in the abstract model will not be able to lead us to the cause of a failure.

3.2. Modeling partial failures

Components across different types of systems have widely varying granularities. Some components, like AS's, are by themselves large distributed systems while blocks of code in a software program may be quite simple. In general, we can associate a component with one or more functionalities, where the number of functionalities is dependent on the system and scale of the component.

We define a *partial failure* of a component to be the case when one or more functionalities of a component fail (or are modified) while the others are unaffected. In this model, we implicitly assume that partial failures within one component are *independent* and do not influence partial failures in other components. While this assumption does not account for more complex failures, we extend our model to take into account simultaneous correlated failures, as well as failures caused by component interactions, in Section 5.

The failure of a functionality within a component manifests itself externally by causing any quark that uses that functionality to fail. To account for this, we use a simple probabilistic model for a partial failure of a component: Given a component, C_i, let probability p_i represent the failure probability of a quark that utilizes component C_i. This probabilistic model makes no assumptions about the specific functionalities that are associated with a component. While in certain applications, one may be able to specify all the functionalities associated with a component, here, we assume that this information is not available. It is important to note that the value p_i is dependent on the distribution of quarks using the failed and successful functionalities of a component C_i ($p_i = 0$, if an unused functionality in a component has failed).

3.3. Basic problem definition

Consider a large scale system with a set of components, $S = \{C_1, C_2, \dots C_n\}$. An instrumentation box monitors some of the quarks of the system where each quark is a tuple $Q = (Q_s, Q_h)$ where Q_s is a set of components and Q_h is a binary health result represented as 0 or 1. Based on this probabilistic model of partial failures and the assumption that partial failures are independent, we define two versions of the root cause localization problem:

Deterministic version: Given several quarks in the system of which some failed (*i.e.,* $Q_h = 0$), determine the potential set of components that have experienced a partial failure *i.e.,* list of components C_i with $p_i > 0$.

Statistical version: Given several quarks in the system of which some failed (*i.e.,* $Q_h = 0$), estimate the partial failure probability p_i for each component C_i.

We refer to the deterministic version as the *partial failure identification* problem. Identifying the components that may have failed in general is easier than estimating p_i and also requires a much smaller statistical sample set of quarks.

4. Potential solutions

In this section, we describe three varied approaches to handling root cause localization in different application domains. While none of these approaches completely solve the general problem, we can use them to show how existing solutions can be applied to our abstract model of root cause localization. This immediately opens up the possibility of applying each solution to a broader set of systems outside their original domain. Of these solutions, the minimum set-cover algorithm addresses the deterministic version of the problem and decision tree learning and logistic regression address the statistical version of the problem. This list of solutions is by no means complete. Other statistical techniques, such as Cohen *et al.*'s approach for automated diagnosis based on Bayesian networks, may also be mapped to our abstract model [8].

4.1. Generic challenges

There are two core challenges in root cause localization, both related to the quantity and quality of our observations of the system.

Component Visibility and System Structure: The accuracy with which we can address the root cause localization problem is dependent on the *coverage* of the quarks (how well our observed quarks cover the components in the system) and the *variety* of the quarks (how much the quarks' associated component sets differ from one another). In practice, the set of components that a quark covers is largely dictated by the system's structure.[1] In particular, given the limitations of end-to-end failures, we cannot localize a fault in a component which is not used by any quarks; nor can we diagnose a failure in a system where all quarks are identical, such as a parallel computing system where every calculation depends on every component in the system. In less extreme cases, we may able to localize a problem to a subset of components but not pinpoint the specific component whose failure triggered a failed quark.

Time granularity: While failures may tend to persist for long periods in certain applications (*e.g.,* bugs in software programs), several applications like Internet routing use inbuilt mechanisms to adapt and recover from failures. To accurately pinpoint component failures that occur for short-periods, we require a significant sample set of quarks that provide good coverage and variety during the period of the failure.

4.2. Minimum set cover

A complete failure model represents a specific case of partial failures where $p = 0$ or $p = 1$. Under this assumption, one can view the root cause localization problem as an optimization problem to identify the *minimum set of failed components* that can explain all the failed quarks. One can transform this optimization problem to the classical *set cover* problem [12]. Determining the *minimum set cover* that covers all failed quarks is equivalent to determining the minimum set of failed components that can explain the root cause of all failed quarks.

The minimum set cover method works only under certain assumptions. First, it assumes a complete failure model where the failure of a component completely lasts during the period of observation *i.e.,* no component recovers from a failure during the observation period. Second, the solution to minimum set cover is not unique. For example, if two components occur in all failed quarks, then the algorithm should report both as suspect as opposed to just one of them.

4.3. Decision tree learning

One statistical approach that can be directly extended to the abstract model is decision tree learning, a technique we have used in prior work to localize failures in Internet service clusters [6, 13]. A *decision tree* is a data structure that represents a classification function, where each branch of the tree is a test on some attribute of the input, and where the leaves of the tree hold the result of the function. *Decision tree learning* is the process of building a decision tree to most accurately classify a set of training data [17].

To solve the root cause localization problem, we learn a decision tree to classify (predict) whether a quark is a success or a failure based on its associated components. Of course, we already know the health of the request—what interests us is the structure of the learned decision tree; looking at which components are used as tests within the decision tree function tells us which components are correlated with request failures. Similarly, by applying decision trees to classify quarks based on their associated components, we can hope to localize faults in our abstract model. Decision tree learning is robust to multiple simultaneous independent faults and gracefully tolerates inconsistencies in the training data.

4.4. Logistic regression

Another statistical technique that can be extended to the root cause localization problem is logistic regression, a technique used by Liblit *et al.* to discover which low-level behaviors in a program's code are most correlated with buggy program runs [14]. Logistic regression fits a linear model to a set of training data, trying to learn a linear function of the low-level code behaviors that will correctly classify a program run as either correct or buggy. Assuming that most of the code behaviors will not be relevant to a failure, Liblit *et al.* regularize the input parameters to force the linear model to use only the few behaviors that correctly characterize the failure. Like decision trees, logistic regression gracefully degrades in the face of inconsistencies. However, it does not handle multiple independent faults well.

4.5. Solutions recap

In this section, we have discussed solutions to the computational and theoretical aspect of the root cause localization problem. While none of the these solutions completely address the abstract problem, they do highlight the potential for sharing ideas and solutions among the various research communities. In particular, this also highlights the potential for theory and statistical experts to provide improved solutions for the general problem.

Of course, the system-specific parts of root cause localization, such as determining exactly what constitutes a failure in a domain, are also important. Additionally, there is a significant opportunity for system-specific techniques to help mitigate the challenges of component visibility, system structure and time granularity. Adding new observation points within the system may increase the visibility of quarks and perhaps increase their coverage and variety. In some systems, it is possible to artificially inject new quarks

[1] Most systems do not provide the flexibility to define quarks with an arbitrary set of components but rather constrain this set based on the system structure.

to probe the system and control the set of components in this quark. This opens up a new class of solutions to root cause localization, not detailed here, based on methodical exploration of possible fault propagation paths. To improve our time granularity in systems where collecting observations exacts some cost, we might be willing to pay that cost to observe more quarks once we notice a fault in the system. Coercing transient failures into longer faults can also lengthen the amount of time we have to collect a significant set of quarks.

5. Refining the Basic Model

The model of a partial failure, as we defined in Section 3.2 is simplistic, in that it assumes failures across components to be independent of each other. In this section, we relax this assumption and refine our model to support two types of complex failures: (a) a common problem simultaneously causes several components to fail; (b) the interaction between a set of components triggers a failure. While by no means are these refinements complete enough to capture various forms of failures, we describe them primarily to show how one can extend our problem to model different types of complex failures.

5.1. Modeling simultaneous correlated failures

There exists many types of systems where several components may simultaneously fail due to an underlying common cause.[2] Such types of failures are typically hard to model and localize without additional knowledge about possible causes of simultaneous failures. For this purpose, we define *attributes* of a component as additional descriptions specified by the underlying system about the potential causes of component failures. *E.g.,* in Internet services, one useful set of attributes might include the operating systems, middleware and versions of each component. In BGP, Caesar *et al.* [3] classify causes of routing events into *disjoint equivalence classes* where each class can be viewed as an attribute. In general, we expect several attributes to be common across different components to capture commonality in failures.

With this refinement, the root cause localization problem boils down to determining the set of components and attributes which appear to be triggering failures in quarks. Many of the statistical learning theory techniques, including decision trees and logistic regression, can be extended to model attributes in conjunction with components. Of particular interest is the case of large-scale homogeneous systems where the functionalities across several components are alike (*e.g.,* nodes in a structured peer-to-peer network) and hence the attributes of many components are alike. Here, when applying statistical techniques, one can model attributes as being equivalent to the components in the system to determine common problems across components.

5.2. Failures caused by component interactions

Some failures are caused not by faults in a single component, but by multiple components interacting together. For example, latent faults in two components and subtle incompatibilities due to version differences can cause otherwise perfectly functioning components to fail when used together. Formally, we define a set of components to have an *interaction-failure* if any quark that uses all these components always fails while any quark that uses only a subset of these components is always successful. This set of components represents the smallest set of components that have an interaction failure. Finding this smallest set of components defines the root cause localization problem in the context of interaction-failures.

While the potential number of interaction failures is exponentially large, two specific constraints in the context of many real-world systems make this problem relatively tractable. First, the number of components involved in an interaction fault is often relatively small. Secondly, the system structure and observed interactions between components limits the possible interaction failures we have to consider.

Theoretically, this problem turns out to be similar to a problem in bio-informatics where the interaction between different genetic abnormalities[3] can potentially be a source of cancerous tumor cells. In [16], Michael Newton proposes a set of statistical techniques to identify these genetic abnormalities and these techniques are potentially applicable in the context of interaction failures.

6. Consequences

The primary goal of this paper is to develop an abstract model that can capture the commonalities among the root cause localization problem in different systems. Part of the hope behind this abstraction is to stimulate discussion among the communities that build these systems and elicit the help of theory and statistical experts in solving the underlying problem.

Abstract modeling also has important ramifications on the design and diagnosis of large-scale systems. From the design perspective, one can explicitly modularize components around failure boundaries, add instrumentation to increase the visibility of quarks, and randomize quark behavior to increase their coverage and variety. From the diagnosis perspective, the solvability of the root cause localization problem for a given system dictates the ease of fault

[2] For example, the spread of the SQL Slammer worm triggered several routers to simultaneously reset.

[3] regions of DNA which have be deleted, amplified or modified

diagnosis. For example, based on the structure of quarks in Internet routing, we can argue that certain types of failures are very hard to diagnose [3].

In practice, not all systems or failures fit the component-quark model. Some systems may inherently not be separable into components because: (a) the system structure inherently is not modular; (b) failures may manifest and corrupt the entire system as opposed to specific parts of the system; (c) end-to-end failures may convey little information about system behavior.

Overall, we hope that this abstract modeling can improve our understanding of large-scale systems and thereby lead to the design of systems that are easier to manage, repair and are more reliable.

References

[1] Tealeaf Technology, Integritea, 2002. http://www.tealeaf.com/.

[2] A. Brown, G. Kar, and A. Keller. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. In *IFIP/IEEE International Symposium on Integrated Network Management*, 2001.

[3] M. Caesar, L. Subramanian, and R. H. Katz. Root cause analysis of Internet routing dynamics. Technical report, U.C. Berkeley UCB/CSD-04-1302, 2003.

[4] D.-F. Chang, R. Govindan, and J. Heidemann. The temporal and toplogical characteristics of BGP path changes. In *ICNP*, 2003.

[5] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer. A statistical learning approach to failure diagnosis. In *ICAC*, May 2004.

[6] M. Y. Chen, A. Accardi, E. Kıcıman, D. Patterson, A. Fox, and E. Brewer. Path-Based Failure and Evolution Management. In *USENIX/ACM NSDI*, 2004.

[7] R. Chillarege. Self-testing software probe system for failure detection and diagnosis. In *Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research*, page 10. IBM Press, 1994.

[8] I. Cohen, M. Goldszmidt, T. Kelly, S. Julie, and J. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. *Proc of OSDI*, 2004.

[9] A. Feldmann, O.Maennel, Z. Mao, A. Berger, and B.Maggs. Locating internet routing instabilities. *ACM SIGCOMM*, 2004.

[10] J. Gray. Why do computers stop and what can be done about it? In *In Proceedings of 5th Symposium on Reliability in Distributed Software and Database Systems*, Los Angeles, CA, 1986.

[11] B. Gruschke. Integrated event management: Event correlation using dependency graphs. In *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 98)*, 1998.

[12] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 1974.

[13] E. Kıcıman and A. Fox. Detecting Application-Level Failures in Component-based Internet Services. *To appear in IEEE Transactions on Neural Networks: Special Issue on Adaptive Systems*, 2005.

[14] B. Liblit, A. Aiken, A. X. Zheng, and M. I. Jordan. Bug Isolation via Remote Program Sampling. In *ACM PLDI*, 2003.

[15] J. Markoff and G. P. Zachary. In Searching the Web, Google Finds Riches. New York Times, Aug 13, 2003.

[16] M. A. Newton. Discovering combinations of genomic alterations associated with cancer. *Journal of the American Statistical Association*, 2002.

[17] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[18] R. Redstone, M. M. Swift, and B. N. Bershad. Using Computers to Diagnose Computer Problems. In *HotOS*, Kauai, HI, 2002.

[19] RIPE's Routing Information Service Raw Data Page. http://data.ris.ripe.net/.

[20] M. Steinder and A. Sethi. Increasing robustness of fault localization through analysis of lost, spurious and positive symptoms. In *Proceedings of IEEE INFOCOM*, 2002.

[21] University of Oregon RouteViews project. http://www.routeviews.org/.