Using Correlated Surprise to Infer Shared Influence

Adam J. Oliner, Ashutosh V. Kulkarni, and Alex Aiken
Stanford University*
Department of Computer Science
{oliner, ashutosh.kulkarni, aiken}@cs.stanford.edu

Abstract

We propose a method for identifying the sources of problems in complex production systems where, due to the prohibitive costs of instrumentation, the data available for analysis may be noisy or incomplete. In particular, we may not have complete knowledge of all components and their interactions. We define influences *as a class of component interactions that includes direct communication and resource contention. Our method infers the influences among components in a system by looking for pairs of components with time-correlated anomalous behavior. We summarize the strength and directionality of shared influences using a Structure-of-Influence Graph (SIG). This paper explains how to construct a SIG and use it to isolate system misbehavior, and presents both simulations and in-depth case studies with two autonomous vehicles and a 9024-node production supercomputer.*

1 Introduction

Consider a complex production system in which something goes wrong: a performance glitch, a strange result, or an outright crash. How might we identify the source of the problem? A fundamental difficulty is that the costs of instrumentation in production systems are often prohibitive. Significant systems are invariably constructed from many interacting subsystems, and we cannot expect to have measurements from every component. In fact, in many systems we will not even know of all the components or of the interactions among the components we do know. This paper is about analyzing systems as they are, generating a potentially partial diagnosis from whatever data is available.

Our method requires only that some of the *components* in the system are instrumented to generate timestamped measurements of their behavior. The type of measurements may depend on the type of component (e.g., a laser sensor may be instrumented differently than a hard disk). Thus, we need a way to compare measurements of different components in a uniform way. We address this issue, and the related question of how to summarize different kinds of measurements from a single component, by mapping all components' behavior to a single dimension: surprise. That is, our method quantifies how anomalous individual component behavior is, as an *anomaly signal*, using deviation from a model of normal component behavior. An important feature of our anomaly signals is that they are real-valued, meaning that the degree to which a component's behavior is anomalous is retained, rather than the common approach of discretizing behavior into "normal" and "abnormal".

When two anomaly signals are correlated, meaning that two components tend to exhibit surprising behavior around the same time, we say that the components share an *influence*. This correlation can arise from a number of interactions, including direct communication and contention for a shared resource. Not all interactions are instantaneous, so we use effect delays—how long it tends to take an anomaly in one component to manifest itself in another—to establish directionality. Correlation is a pairwise relationship and delay is directional, so the most natural structure to summarize influence is a graph. A Structure-of-Influence Graph (SIG) encodes strong influence as an edge between components, with optional directionality to represent a delay.

Passively collected data, if devoid of hints like "component A sent a message to component B," cannot be used to infer causality: the strongest possible mathematical statement is that the behavior of one component is correlated with another. An advantage of using statistical correlation is that it enables asking "what-if" queries, after the fact. For example, it is easy to add a new "component" whose anomaly signal is large around the time bad behavior was observed. Other, real, components that share influence with the synthetic component are likely candidates for contributors to the problem.

Our goal is to generate a structure, informed by models of component behavior, that enables a user to more easily answer prediction and diagnosis questions. The SIG method has several desirable properties:

*This work was supported in part by NSF grant CCF-0915766 and the DOE High-Performance Computer Science Fellowship.

- Building a SIG requires no intrusive instrumentation; no expert knowledge of the components; and no knowledge about communication channels (e.g., the destination of a message), shared resources, or message content. Our method is passive and can treat components as black boxes.
- Influence describes correlation, not causality. A key feature of our approach is to drop the assumption that we can observe all component interactions and focus on the correlations among behaviors we can observe (see Section 2).
- By working directly with a real-valued, rather than binary, anomaly signal, our method degrades gracefully when data is noisy or incomplete.
- Our experimental results show that SIGs can detect influence in complex systems that exhibit resource contention, loops and bidirectional influence, time-delayed effects, and asynchronous communication.

In this paper, we present the SIG method and work through an example (Section 3); perform several controlled experiments using a simulator (Section 4) to explore parameters like message drop rate, timing noise, and number of intermediate components; describe the central case study of the paper, how we took passively collected measurements from two autonomous vehicles and built SIGs that enabled us to identify the source of a critical bug (Section 5); and briefly present a significantly different second example by isolating a bug in a production supercomputer (Section 6).

2 Related Work

There is an extensive body of work on system modeling, especially on inferring the causal or dependency structure of distributed systems. Our method distinguishes itself from previous work in various ways, but primarily in that we look for *influences* rather than *dependencies*.

Dependency graphs, or some probabilistic variant (e.g., Bayesian networks), are frequently proposed for prediction and diagnosis of computer systems. There have been a number of recent attempts at dependency modeling in distributed systems. Pinpoint [7, 8] and Magpie [3] track communication dependencies with the aim of isolating the root cause of misbehavior; they require instrumentation of the application to tag client requests. Pip [17] aims to infer causal paths and requires an explicit specification of the expected behavior of a system. In order to determine the causal relationships among messages, Project5 [1] and WAP5 [18] use message traces and compute dependency paths. All of these projects compute dependencies, and therefore cannot deal well with missing dependency information or resource contention.

Much of this dependency modeling work requires that the system be actively perturbed by instrumentation or by probing [5, 6, 9, 10, 19]. Unfortunately, for many important systems, no such modifications are possible (for reasons of performance, administration, or cost).

Shrink [11] and SCORE [12] look for the root cause of faults in wide-area networks using a two-level graph. Shrink weights dependencies based on likeliness estimates. SCORE looks for shared risk, which measures how strongly correlated failures are across hosts. Finally, recent work by Bahl [2] aims to infer multi-level dependency graphs that model load-balancing and redundancy.

With few exceptions [4], in previous work events are intrinsically binary (i.e., happen or not). Our approach, which abstracts components as a real-valued signal, retains strictly more information about component behavior.

As systems grow in scale, the sparsity of instrumentation and complexity of interactions will only increase. Our method infers a broad class of interactions using, typically, fewer assumptions about available data than previous work.

3 The Method

This section describes how to construct and interpret a Structure-of-Influence Graph (SIG). The construction process consists of four steps: decide what information to use from each component (Section 3.1), measure the system's behavior during actual operation as *anomaly signals* (Section 3.2), compute the pairwise cross-correlation between all components' anomaly signals to determine the strength and delay of correlations (Section 3.3), and construct a SIG where the nodes are components and edges represent the strength and delay of correlations between components (Section 3.4). We later apply these techniques to idealized systems (Section 4) and real systems (Sections 5 and 6).

3.1 Modeling

The choice of component models determines the semantics of the anomaly signal and, consequently, of the SIG. For example, if we model a program using the distribution of system call sequences and model a memory chip using ECC errors, then the relationship of these components in the resulting SIG represents how strongly memory corruption influences program behavior, and vice versa. There is not, therefore, a single correct choice of models; for a particular question, however, some models will produce SIGs better suited to providing an answer.

We have found two models particularly useful in practice: one based on message timing, which is useful for systems where timing behavior is important (e.g., embedded systems) and at least some classes of events are thoroughly logged (see Section 5), and one based on the information

content of message terms, useful for systems where logging is highly selective and ad hoc (see Section 6). The timing model keeps track of past interarrival times (timestamp first-differences) and computes how "surprising" the most recent spacing of messages is (see Section 3.2.1); the term entropy model looks at the distributions of message contents using an existing method [15].

3.2 Anomaly Signal

We quantify the behavior of components in terms of surprise: the *anomaly signal* $\Lambda_j(t)$ describes the extent to which the behavior of component j is anomalous at time t. The instantaneous value of the signal is called the *anomaly score*. Let $\Lambda(t) = 0$ for any t outside the domain of the anomaly signal. We require that $\Lambda_j(t)$ has finite mean μ_j and standard deviation σ_j.

The anomaly signal should usually take values close to the mean of its distribution—this is an obvious consequence of its intended semantics. The distance from the mean corresponds to the extent to which the behavior is anomalous, so values far from the mean are more surprising than those close to the mean.

The user defines what constitutes surprising behavior by selecting an appropriate model. For example, one could use deviation from average log message rate, degrees above a threshold temperature, the divergence of a distribution of factors from an expected distribution, or some other function of measurable, relevant signals.

3.2.1 Computing the Anomaly Signal

In this section, we discuss the mechanics of computing the anomaly signal $\Lambda_j(t)$ for the timing model mentioned in Section 3.1. We describe the offline version.

Let S be a discrete signal from some component, consisting of a series of time (non-decreasing timestamp) and value pairs: $S = \langle (t_0, v_0), (t_1, v_1), \ldots, (t_s, v_s) \rangle$.

Individually, denote $S(i) = (t_i, v_i)$, $T(i) = t_i$, and $V(i) = v_i$. This work gives special attention to the case when $V(i)$ is the first difference of the time stamps (interarrival times): $V(i) = T(i) - T(i-1)$ and $V(0) = \phi$ (null).

To compute anomaly signals, we compare a histogram of a recent window of behavior to the entire history of behavior for a component. Let h be the (automatically) selected bin width for the histogram (in seconds), let w be the size of the recent history window in number of samples, and let $k = \lceil \frac{\max v_i - \min v_i}{h} \rceil$ be the number of bins. For each bin $H(j)$ in the historical histogram, count the number of observations $V(i)$ such that $jh \le V(i) < jh+1$. Let $R(T(i))$ be the analogous histogram computed from the previous w samples, ending with $V(i)$. Note that $R(T(i))$ is not defined for the samples $V(1)$ through $V(w)$. Let H' and $R'(t)$ be the corresponding probability distributions, where the count in each bin is divided by the total mass of the histogram; H has a mass of $s-1$ and $R(t)$ has a mass of w.

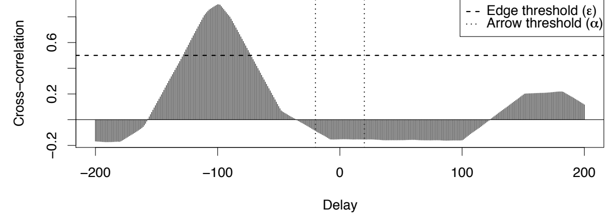

Figure 1. The normalized cross-correlation between components A and B.

Compute the Kullback-Leibler divergence [13] between each recent distribution $R'(t)$ and the historical distribution H', producing the anomaly signal $\Lambda(t)$:

$$\Lambda(t) = D_{KL}(R'(t)||H') = \sum_{k \in R'(t)} R'(t,k) \log_2 \frac{R'(t,k)}{H'(k)}.$$

Intuitively, KL-divergence is a weighted average of how much the fraction of measurements in bin $R'(t,k)$ differs from the expected fraction $H(k)$.

After computing $\Lambda_j(t)$ for each component, we store the sampled signals as an $n \times m$ matrix, where n is the number of components and m is the number of equi-spaced times at which we sample each anomaly signal. We then process these matrices as described starting in Section 3.3. Observe that, having represented the system as a set of anomaly signals, the rest of our method is system-independent.

3.3 Correlation and Delay

For each pair of components (i,j), compute the normalized cross-correlation of their anomaly signals:

$$(\Lambda_i \star \Lambda_j)(t) \equiv \int_{-\infty}^{\infty} \frac{[\Lambda_i(\tau) - \mu_i][\Lambda_j(t+\tau) - \mu_j]}{\sigma_i \sigma_j} d\tau. \quad (1)$$

The function $(\Lambda_i \star \Lambda_j)(t)$ gives the Pearson product-moment correlation coefficient of the anomaly signals of components i and j, with an offset, or *delay*, of t time steps; it is the correlation of the two signals if Λ_i were delayed relative to Λ_j by t. Consider two hypothetical components, A and B, whose cross-correlation is plotted in Figure 1. There is a peak at $t = -100$ because anomalies on A tend to appear 100 units of time before they do on B. This plot would be represented in the SIG by an edge $A \to B$.

The resulting n^2 cross-correlation functions of an n-component system are the primary output of our analysis. (It is worth noting that Project5 uses a form of signal correlation with communication events to compute dependencies [1].) In general, however, these correlation vectors contain

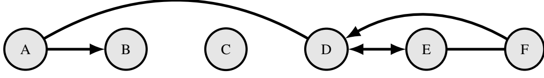

Figure 2. The Structure-of-Influence Graph for a system that includes A and B.

too much information to present a useful view of a system; we distill these data into simpler forms. First, we represent two salient features of the functions (specific extrema values and positions) as two order $n \times n$ matrices: $\mathbf{C}$ and $\mathbf{D}$. Second, in Section 3.4, we transform these matrices into a SIG. The SIG is often the shortest path to insights about a system, but the underlying data is always available for inspection or further analysis.

We now construct the correlation matrix $\mathbf{C}$ and delay matrix $\mathbf{D}$ from the cross-correlation functions. Consider a particular pair of components, i and j. Let d_{ij}^- and d_{ij}^+ be the offsets closest to zero, on either side, at which the cross-correlation function is most extreme:

$$d_{ij}^- = \max(\text{argmax}_{t \leq 0}(|(\Lambda_i \star \Lambda_j)(t)|)) \text{ and}$$
$$d_{ij}^+ = \min(\text{argmax}_{t \geq 0}(|(\Lambda_i \star \Lambda_j)(t)|)),$$

where $\text{argmax}_t f(t)$ is the set of t-values at which $f(t)$ is maximal. (One could also select the delays furthest from zero, if that is more appropriate for the system under study.) Next, let c_{ij}^- and c_{ij}^+ be the correlations observed at those extrema: $c_{ij}^- = (\Lambda_i \star \Lambda_j)(d_{ij}^-)$ and $c_{ij}^+ = (\Lambda_i \star \Lambda_j)(d_{ij}^+)$.

Let entry $\mathbf{C}_{ij}$ of the correlation matrix be c_{ij}^- and let $\mathbf{C}_{ji}$ be c_{ij}^+. (Notice that $c_{ij}^+ = c_{ji}^-$.) Similarly, let entry $\mathbf{D}_{ij}$ of the delay matrix be d_{ij}^- and let $\mathbf{D}_{ji}$ be d_{ij}^+.

3.4 Structure-of-Influence Graph (SIG)

A Structure-of-Influence Graph (SIG) is a graph $G = (V, E)$ with one vertex per component and edges that represent influences. Edges may be undirected, directed, or even bidirectional, to indicate the delay(s) associated with this influence. This section explains how to construct a SIG.

Consider the $n \times n$ matrices $\mathbf{C}$ and $\mathbf{D}$. There is an edge between i and j if $\max(|\mathbf{C}_{ij}|, |\mathbf{C}_{ji}|) > \varepsilon$. Let α be the threshold for making an edge directed; the type of edge is determined as follows:

$$((|\mathbf{C}_{ij}| > \varepsilon) \Rightarrow (\mathbf{D}_{ij} > -\alpha)) \wedge ((|\mathbf{C}_{ji}| > \varepsilon) \Rightarrow (\mathbf{D}_{ji} < \alpha)) \Rightarrow i - j;$$
$$(|\mathbf{C}_{ij}| > \varepsilon) \wedge (\mathbf{D}_{ij} < -\alpha) \Rightarrow i \rightarrow j;$$
$$(|\mathbf{C}_{ji}| > \varepsilon) \wedge (\mathbf{D}_{ji} > \alpha) \Rightarrow i \leftarrow j;$$
$$(|\mathbf{C}_{ij}| > \varepsilon) \wedge (\mathbf{D}_{ij} < -\alpha) \wedge (|\mathbf{C}_{ji}| > \varepsilon) \wedge (\mathbf{D}_{ji} > \alpha) \Rightarrow i \leftrightarrow j.$$

The time complexity of our method on a system with n components, given an algorithm to compute cross-correlation in time $O(m)$, is $O(n^2 m)$. For large systems, we may wish to compute only a subset of the SIG: all influences involving a set of $n' << n$ components. This is equivalent to filling in only specific rows and columns of $\mathbf{C}$ and $\mathbf{D}$ and requires time $O(nn'm)$.

Recall our example components A and B. Using the cross-correlation of A and B, shown in Figure 1, we apply the thresholds $\alpha = 20$ and $\varepsilon = 0.5$ and plot a SIG. Although $|\mathbf{C}_{ji}| < \varepsilon$, we have $|\mathbf{C}_{ij}| = 100 > \varepsilon$, so there will be an edge between A and B. Furthermore, $\mathbf{D}_{ij} < -\alpha$, so the edge will be directed: $A \rightarrow B$. Figure 2 gives a SIG for a hypothetical system that includes A and B as components. In subsequent sections we discuss how the values of α and ε can be chosen or set automatically.

3.5 Interpreting a SIG

An edge in a SIG represents a strong ($> \varepsilon$) influence between two components. The absence of an edge does not imply the absence of a shared influence, merely that the anomalies identified by the models are not strongly correlated—a different choice of models may yield a different graph. More specific interpretations arise from understanding particular models and underlying components.

A directed edge implies that an anomaly on the source component (tail of the arrow) tends to be followed shortly thereafter by an anomaly on the sink component (head of the arrow). Bidirectional edges mean that influence was observed in both directions, which may mean either that the influence truly flows in both directions or that it is unclear which directionality to assign (this situation can arise with periodic anomalies). An undirected edge means that, to within a threshold α, the anomalies appear to occur simultaneously. This happens, for instance, when a mutually influential component is causing the anomalies. Such shared components sometimes introduce cliques into the SIG.

When used for problem isolation, the most important piece of actionable information provided by our method is a concise description, in the form of graph edges, of which components seem to be involved. Further, the strength and directionality on those edges tell the order in which to investigate those components. In a system of black boxes, which is our model, this is the most any method can provide.

4 Controlled Experiments

In this section, we study the notion of influence and our method for computing it under a variety of adverse conditions: measurement noise, message loss, and tainted training data. We use simulation experiments on idealized systems consisting of linear chains of components. We use chains so the results of our simulations are easy to interpret; our method is not limited to chains, and our experiments with real systems in subsequent sections involve much more complex structure. Our goal is to thoroughly examine a specific question: Given a known channel of data and resource

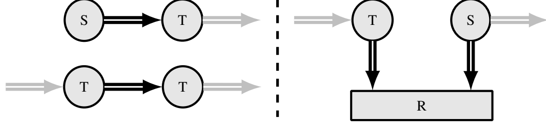

Figure 3. The three basic structures in our simulated systems, built from sources ("S"), tasks ("T"), and resources ("R").

interactions through which influence could propagate, what is the strength (ε) of the influence inferred by our method? Our results show that, for many realistic circumstances, influence can propagate through long chains of components, and we can detect it.

4.1 System Components

Our simulations use three types of components: *sources*, which generate data, *tasks*, which process data, and *resources*, which are required by sources and tasks to perform these actions. Pairs of sources and tasks, shown in Figure 3, can influence each other via direct communication or via competition over a shared resource. We study linear chains of such structures in which the first component is a source; that source, called the *head*, is designed to sometimes misbehave during the experiments and acts as the root cause that we wish to find. The only input to our method is a pair of timestamp vectors (one for the *head* of the chain and one for the *tail*) corresponding to message sending times. No information about simulation parameters or intermediate components is provided.

Influence can flow over a direct source-to-task or task-to-task channel either by *timing* (anomalous input timing may cause anomalous output timing, as in a producer-consumer interaction), by *semantics* (tasks may take more or less time to process uncommon messages), or *both*. Influence can flow over a shared resource only through timing (e.g., the degree of contention may influence timing); we do not simulate implicit communication through shared memory.

4.2 Component Behavior

We characterize timing behavior of components by distributions of interarrival times, which is sufficient to compute anomaly signals (see Section 3.2.1). These experiments use Gaussian (normal) distributions. Let η_x denote a normally distributed random variable with mean 1 and standard deviation σ_x. Fixing the mean makes the problem more difficult, because abnormal behavior does not result in consistently more or fewer messages (merely greater variance) and because anomalous behavior looks like measurement imprecision (noise).

A *source* generates the message `0` every η_n seconds. A source may be *blocked* if any downstream component is not *ready* to receive the message, in which case it waits until all such components are ready, *sends* the message, and then waits η_n seconds before trying to generate the next message. We consider three types of anomalous behavior at the head node: *timing* (generates a message every η_a seconds), *semantics* (generates the message `1`), and *both*.

A *task* begins processing a message upon receipt, taking η_0 seconds for a `0` message and η_1 seconds for a `1` message. After processing, a task sends the same message to the next component downstream. If that component is not ready, the task is blocked. A task is ready when it is not blocked and not processing a message.

A *resource* receives and processes messages; it can simultaneously process up to R messages for some capacity R. A resource requires η_r seconds to process a message and is ready to receive whenever it is processing fewer than R messages. Resources service simultaneous requests in a random order.

When the head or tail sends a message, as described above, it records the time at which the message was sent; our method computes the influence given only this pair of timestamp lists. While real systems may exhibit more complex behavior, these simple rules are enough to capture different classes of inputs (`0` vs. `1` messages), resource contention, and potential timing dependencies.

4.3 Methodology

Each experiment, resulting in a single influence value ε, involves two independent simulations of a chain over a time period long enough for the head to send 10,000 messages. The first simulation yields a trace that is used for training (a behavior baseline), and the second, a monitoring trace, is used to build the SIG. Except where otherwise indicated, the training trace does not contain anomalous behavior, and the monitoring trace contains a contiguous period of anomalous behavior lasting 5% of the trace (500 messages).

Resources have exactly two connected components, each with a normal average message sending rate of 1 per second, so every resource has a capacity of $R = 2$. That is, there should be little contention during normal operation. The number of resources is denoted by `#R=?`; resources are evenly distributed along the chain. Except where otherwise noted, $\sigma_n = \sigma_r = \sigma_0 = 0.01$ and $\sigma_a = \sigma_1 = 0.1$. For the component model, the histogram bin size is $h = 0.01$ seconds (set automatically) and the window size is $w = 500$ samples (chosen to match the anomalous period).

4.4 Experiments

Baseline: We compute an influence strength baseline for each simulation that represents the expected correlation of anomaly signals if the head and tail were independent. For

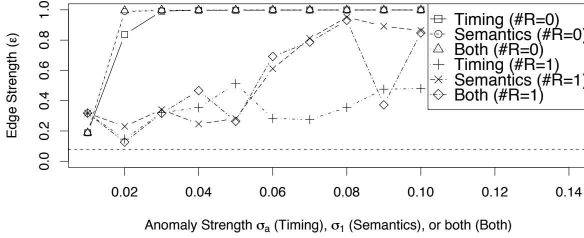

Figure 4. Behavior of the basic components: a single task (#R=0) and a single resource (#R=1), each with a driving source head.

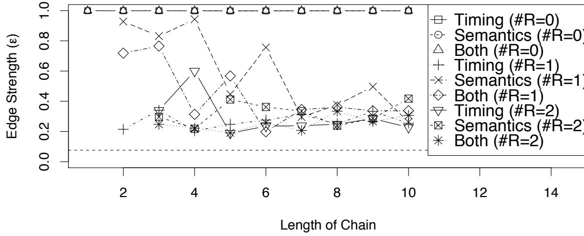

Figure 5. Contention carries timing influence across resources and tasks pass along semantic influence, even down long chains.

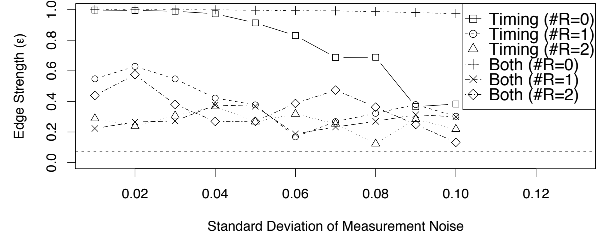

Figure 6. Our method degrades gracefully when timing measurements are noisy.

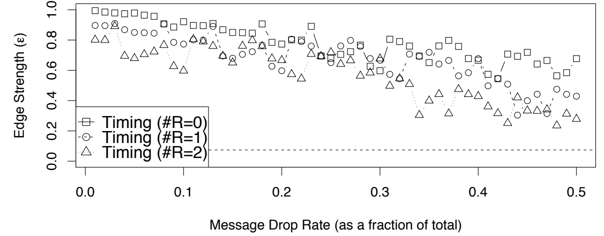

Figure 7. Our method is robust against uniform message loss, even at rates of 50%.

all experiments in this section, that baseline ranges from 0.06 to 0.1; the average for each set of experiments is plotted in the figures as a dashed line. Thus, any ε value above 0.1 can be considered statistically significant. This is comparable to the edge threshold of $\varepsilon = 0.15$ that we use when considering real systems (see Sections 5 and 6).

Basic Components: Figure 4 shows the strength of influence across the basic simulation components of Figure 3 for varying anomaly strengths. Tasks propagate both timing and semantic influence, while resources only propagate timing. Tasks change semantic influence into timing influence, however, which resources can then propagate. Overprovisioned resources do not propagate influence; resources with adequate capacity, which we simulate, propagate timing influence. Note that we detect influence even when anomalous behavior looks similar to normal behavior: even during normal operation there is variation in component behavior, and these variations may be correlated between components.

Length and Composition: When there is more than one component, we find that influence generally fades with increasing chain length, but remains detectable (see Figure 5). When there are no resources, however, message semantics are passed all the way to the tail and the influence is undiminished. For the rest of the section, chains contain six components; this length is long enough to exhibit interesting properties and is comparable to the diameter of the autonomous vehicle graphs in Section 5.

Signal Noise: Our method is robust against noisy data. As we add more and more Gaussian noise to the timing measurements, it obscures some of the influence of anomalous behavior but does not mask it entirely. This is true when noise is added to the resulting measurements (measurement imprecision, as in Figure 6) or to the components (omitted for space, but similar to Figure 6). Note that even "normal" timing variations at the head can influence timing at the tail.

Message Loss: For our timing model, message loss is simply another form of noise that tends to introduce outliers. For example, if a component output messages at t_1, t_2, and t_3, but the second measurement is lost, our timing distribution will erroneously include the value $t_3 - t_1$, which will be twice as large, on average, as most of the other measurements. To make our job more difficult, we simulate the case when our training data has no lost messages but the monitoring data does. Figure 7 shows that our statistical methods are not strongly sensitive to missing data.

Tainted Training: The problem of good training data exists for every anomaly-based method. Figure 8 shows that, as the fraction of training data that includes anomalous behavior increases, influence remains easily detectable. Tainting does not tend to introduce new correlations; existing correlations may appear less significant, as in the middle line. Training data need only be statistically representative, so it can include unusual periods (like startup) or bugs.

These experiments show that influence propagates through systems in a measurable way and that our method can detect this influence under a variety of circumstances. Although the simulations consider a restricted class of systems, the systems we study in Sections 5 and 6 contain far more complex structure, including asynchronous communication through shared memory, high degrees of network

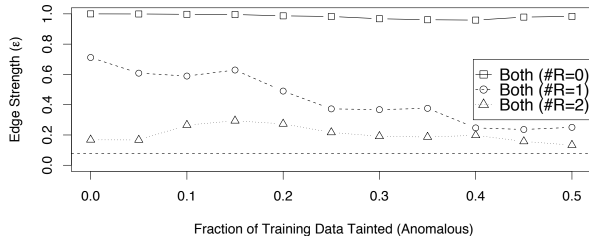

Figure 8. Our ability to detect influence does not depend on collecting clean training data.

fan-in and fan-out, loops and cycles, and potentially multiple sources of anomalous behavior.

5 Stanley and Junior

DARPA launched the Grand Challenge in 2003 to stimulate autonomous vehicle research. The winner of the Grand Challenge was a diesel-powered Volkswagen Touareg R5 named Stanley, an autonomous vehicle developed at Stanford University [21]. Stanford's entry in the successive contest, a modified 2006 Volkswagen Passat wagon named Junior, placed second in the Urban Challenge [14].

Many of the autonomous vehicles' components run in tight loops that output log messages at each iteration. Deviations from normal timing behavior are rare, but, more importantly, we expect the anomalies to correspond with semantically abnormal situations. For example, if the route-planning software takes unusually long to plot a path, the vehicle may be in a rare driving situation (e.g., a 4-way stop where the driver with right-of-way is not proceeding).

5.1 Stanley's Bug

During the Grand Challenge race, Stanley suffered from a bug that manifested itself as unexplained swerving behavior. That is, the vehicle would occasionally veer around a nonexistent obstacle. According to the Stanford Racing Team, "as a result of these errors, Stanley slowed down a number of times between Miles 22 and 35" [21]. The bug forced Stanley off the road on one occasion, nearly losing the race. We explain this bug in more detail in Section 5.6, but, for the time being, let us suppose that all we know is that we were surprised by Stanley's behavior between Miles 22 and 35 of the race and that we would like to use the method described in this paper to find an explanation.

5.2 Experiments

During the Grand Challenge and Urban Challenge races, each vehicle was configured to record inter-process communication (IPC) between software components, including the sensors. These messages were sent to a central logging

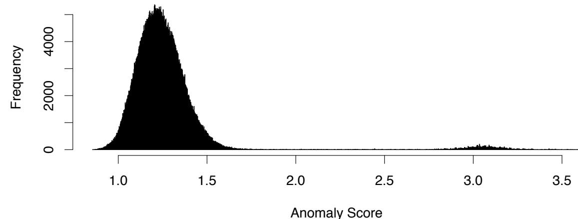

Figure 9. Anomaly signal distribution for Stanley's `GPS_POS` component.

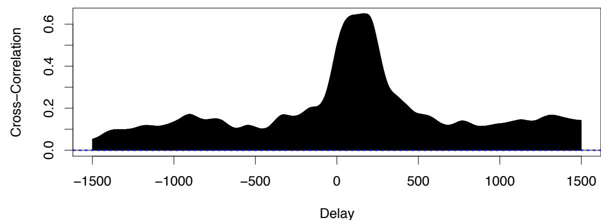

Figure 10. Cross-correlation of Stanley's `PLANNER_TRAJ` and `LASER1` components.

server and written to disk. Only a subset of the components generated logs. The messages indicate their source, but not their destination; the absence of such information means that most previous work would be inapplicable to this data set. We sample the anomaly signals at intervals of 0.04 seconds; this sampling interval was set (automatically) to be the smallest non-zero interarrival time on any single component on Stanley.

Computing a SIG requires only two parameters, ε and α, neither of which need to be fixed *a priori*; adjusting them to explore the impact on the resulting graph can be informative (e.g., if an entire clique becomes disconnected due to a small decrease in ε, we know that the shared influence has roughly the same impact on all members of the clique). For the component model, the histogram bin size(s) h is set automatically using Sturges' formula [20], and we use a recent window size of $w = 100$ samples; our results are not sensitive to this choice (details omitted for space reasons).

5.3 Anomaly Signals

For each component, we use the timing model and computations described in Section 3.2.1 to generate an anomaly signal. A "good" anomaly signal has low variance when measuring a system under typical load, in accordance with its semantics (usual behavior is not surprising behavior). Often, the vehicle components generate normally distributed or exponentially distributed anomaly scores. Sometimes, as in Figure 9, the anomaly scores are bimodal, with one cluster around typical behavior and another cluster around anomalous behavior.

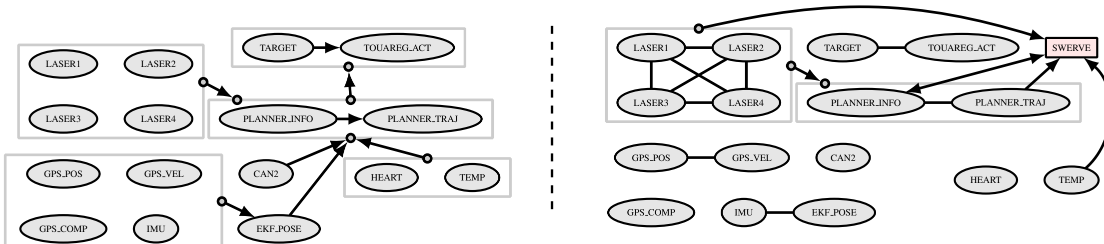

Figure 11. Known dependency structure of Stanley on the left (including only logged components) and the automatically generated SIG on the right (with $\varepsilon = 0.15$ and $\alpha = 90$). The special `SWERVE` **component is explained in Section 5.6.**

5.4 Cross-correlation

We proceed by computing the cross-correlation between all pairs of components within each car using a discrete version of Equation 1. When two components do not share an influence, the cross-correlation tends to be flat. This can also happen when two components share an influence but there is no consistent delay associated with it. When there is a shared influence, we see a peak or valley in the cross-correlation function. The more pronounced the extrema, the stronger the inferred influence. Figure 10 gives the cross-correlation between Stanley's `PLANNER_TRAJ` and `LASER1` components. We see a peak whose magnitude (correlation) exceeds 0.6, which is relatively large for this system. We can already conclude that `PLANNER_TRAJ` and `LASER1` likely share an influence. The strong correlation at a small positive lag means the `LASER1` anomalies tend to precede those on `PLANNER_TRAJ`.

In addition to the magnitude of the correlation, we can learn from the location of an extremum on the delay axis. Here, we see that it occurs at a delay of, roughly, 100–200 samples (at a 0.04-second sampling interval, the delay is around 4–8 seconds). In this case, we are looking at the result of computing $(\texttt{PLANNER_TRAJ} \star \texttt{LASER1})(t)$, so the interpretation is that anomalies on `PLANNER_TRAJ` tend to occur between 4–8 seconds *after* those on `LASER1`. More important than the value, however, is the direction: the laser anomalies precede the planner software anomalies. When isolating the bug mentioned in Section 5.1, this turns out to be an important piece of information.

5.5 SIGs

Using the method described in Section 3.4, we distill the cross-correlation matrices into SIGs. We compute a statistical baseline for Stanley, similar to Section 4.4, of just under 0.15 (details omitted for space reasons). For Stanley, a SIG with $\varepsilon = 0.15$ and $\alpha = 90$ is shown on the right in Figure 11, alongside the software dependency diagram. As a notational shorthand to reduce graph clutter, we introduce grey boxes around sets of vertices. An arrow originating from a box denotes a similar arrow originating from every vertex inside the box; an arrow terminating at a box indicates such an arrow terminating at every enclosed vertex. Consequently, the directed arrow in Stanley's SIG from the box containing `LASER*` to the box containing `PLANNER*` indicates that each laser component shares a time-delayed influence with each planning software component. We explain the `SWERVE` component, plotted as a red rectangle, in Section 5.6. Most of the strongest influences do not map to stated dependencies, meaning the dependency diagram has obscured important interactions that the SIG reveals.

The edges in the dependency diagram indicate intended communication patterns, rather than functional dependencies. In fact, Stanley had five laser sensors, not four, but one broke shortly before the race. The downstream components were clearly not dependent on that laser, in the sense that they continued working. If another laser malfunctioned, would it affect the behavior of the vehicle? The dependency diagram is unhelpful, but in Section 5.6 we show how to query a SIG to elucidate this kind of influence.

Even in an offline context, SIGs are dynamic. By using only a subset of the data, such as from a particular window of time, we can see the structure of influence particular to that period. Furthermore, we can consider a series of such periods to examine changes over time. A SIG for Junior showing influence during the second race mission, relative to the first is plotted in Figure 12. Dashed grey means the edge is gone, thicker edges are new, and an open arrowhead means the arrow changed. Disconnected components are omitted. Although the component models use the entire log as training data, we generate this graph using only data from the first and second thirds of the Urban Challenge (called "missions"), with edges marked to denote changes in structure. Notice that many components are disconnected in this SIG, and thus omitted from the plots, as a consequence of the value of ε. As this parameter increases, edges vanish; as it tends to zero, the graph becomes fully connected.

One change that stands out is the new influence between `RADAR5` and `PASSAT_EXTOUTPUT2`. Further examination of the data shows several anomalies at the radar components; a lower value of ε would have shown the radars in

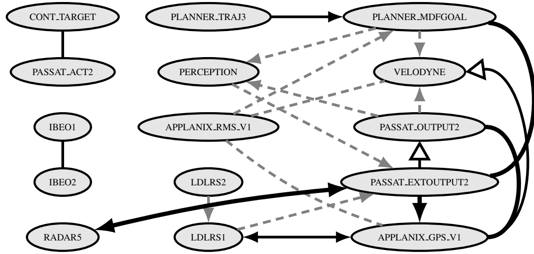

Figure 12. Dynamic changes in Junior's SIG, with $\varepsilon = 0.15$ and $\alpha = 90$.

a clique, but RADAR5 exhibited the most pronounced effect. Many of the edges that disappeared did so because anomalies during the first mission did not manifest in the second. Studying changes in influence structure over time enables us to discover such dynamic behaviors.

For the vehicles, physical connectivity does not change during a race, so any changes are a consequence of which influences we happened to observe during that period. If we only had access to the latter half of the Grand Challenge logs, for instance, it would have been more difficult to diagnose the swerving bug because there was little swerving behavior to observe. For larger systems and longer time periods, changes in the SIG may correspond with upgrades or physical modifications.

5.6 Swerving Bug

Starting only with the logs from the Grand Challenge race and the knowledge that Stanley exhibited strange behavior between Miles 22 and 35, we show how a simple application of our method points directly at the responsible components.

First, we construct a synthetic component called SWERVE, whose anomaly signal is nonzero only for a single period that includes the time the majority of surprising behavior (swerving) was observed. We could refine this signal to indicate more precisely when the bug manifested itself, but the surprising result is that, even with this sloppy specification of the bug, our statistical method is still able to implicate the source of the problem.

Second, we update Stanley's SIG as though SWERVE were just another component. The result is the graph on the right in Figure 11. Consider the correlation values for the seven components with which SWERVE seems to share an influence: all four laser sensors, the two planning software components, and the temperature sensor.

The temperature sensor is actually anti-correlated with SWERVE. This spurious correlation is the price we pay for our sloppy synthetic anomaly signal; it occurs because the SWERVE anomaly signal is (carelessly) non-zero only near the beginning of the race while the TEMP anomaly signal, it turns out, is increasing over the course of the race.

Now there are six components that seem to be related to the swerving, but the SIG highlights two observations that narrow it down further: (i) directed arrows from the lasers to the planner software mean that the laser anomalies preceded the planner anomalies and (ii) the four lasers form a clique, indicating that there may be another component, influencing them all, that we are not modeling. (Recall the discussion in Section 3.5.) Observation (i) is evident from the SIG and Figure 10 confirmed that the planner typically lagged behind the lasers' misbehavior by several seconds. Observation (ii) suggests that a component shared by the lasers, rather than the lasers themselves, may be at fault.

According to the Stanford Racing Team, these observations would have been sufficient to quickly diagnose the swerving bug. At the time, without access to a SIG and working from the dependency diagrams, it took them two months to chase down the root cause. This is a natural consequence of how a dependency diagram is used: start at some component near the bottom of the graph, verify that the code is correct, move up to the parent(s), and repeat. All of the software was working correctly, however, and until they realized there were anomalies at the input end (the opposite end from the swerving) there was little success in making a diagnosis. The SIG would have told them to bypass the rest of the system and look at some component shared by the lasers, which was the true cause.

The bug, which turned out to be a non-deterministic, implicit timing dependency triggered by a buffer shared by the laser sensors, is still not fully understood. However, the information our method provides was sufficient for the Racing Team to avoid these issues in Junior. Indeed, this is evident from Junior's SIG, Figure 12, where the lasers are not influencing the rest of the system as they were in Stanley. More information on the swerving bug can be found in the journal paper concerning Stanley [21].

6 Thunderbird Supercomputer

We briefly describe the use of SIGs to localize a non-performance bug in a non-embedded system (the Thunderbird supercomputer [16]) of significantly larger scale ($n = 9024$) using a component model based on the frequency of terms in log messages [15] instead of timing. Except for the different anomaly signal, the construction and use of the SIGs is identical to the case study in Section 5; we focus only on the new aspects of this study.

Say that Thunderbird's administrator notices that a particular node generates the following message and would like to better understand it: `kernel: Losing some ticks... checking if CPU frequency changed`. For tightly integrated systems like Thunderbird, static dependency diagrams are dense with irrelevant edges; meanwhile, the logs do not contain the information

about component interactions or communication patterns necessary for computing dynamic dependencies.

Instead, we show how a simple application of our SIG method leads to insight about the bug. Using the model based on term frequency mentioned above [15], which incorporates no system-specific knowledge, we first compute anomaly signals for all components. As with Stanley's swerving bug (see Section 5.1), we can easily synthesize another anomaly signal for each node, nonzero only when it generates this "CPU error".

The resulting SIG contains clusters of components (both synthetic and normal) that yield a surprising fact: when one node generates the CPU error, other nodes tend to generate both the CPU error and other rare messages. Furthermore, the administrator can quickly see that these clusters correspond to job scheduling groups, suggesting that a particular workload may be triggering the bug. Indeed, it turns out that there was a bug in the Linux kernel that would cause it to skip interrupts under heavy network activity, leading the kernel to erroneously believe that the clock frequency had changed and to generate the misleading CPU error. The SIG method shows immediately that the error shares an influence with other nodes in the same job scheduling group; an insight that helps both isolate the cause and rule out other avenues of investigation.

7 Contributions

In this paper, we propose using *influence* to characterize the interactions among components in a system and present a method for constructing Structure-of-Influence Graphs (SIGs) that model the strength and temporal ordering of influence. We abstract components as *anomaly signals*, which enables noise-robust modeling of heterogeneous systems. Our simulation experiments and case studies with two autonomous vehicles and a production supercomputer show the benefits of using influence over traditional dependencies and demonstrate how an understanding of influences can equip users to better identify the sources of problems.

Acknowledgments

The authors would like to thank the following people for their time, expertise, and data: Xuân Vũ, Jon Stearley, Peter Hawkins, Randall LaViolette, Mike Montemerlo and the rest of the Stanford Racing Team, and our reviewers.

References

[1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Methitacharoen. Performance debugging for distributed systems of black boxes. In *SOSP*, pages 74–89, 2003.

[2] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *SIGCOMM*, 2007.

[3] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *OSDI*, 2004.

[4] P. C. Bates. Debugging heterogeneous distributed systems using event-based models of behavior. *ACM Transactions on Computer Systems*, 13(1):1–31, 1995.

[5] M. Brodie, I. Rish, and S. Ma. Optimizing probe selection for fault localization. In *Intl. Workshop on Distributed Systems: Operations and Management (DSOM)*, October 2001.

[6] A. Brown, G. Kar, and A. Keller. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. In *IEEE IM*, pages 377–390, Seattle, WA, 2001.

[7] M. Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based failure and evolution management. In *NSDI*, 2004.

[8] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: problem determination in large, dynamic internet services. In *DSN*, June 2002.

[9] S. Chutani and H. Nussbaumer. On the distributed fault diagnosis of computer networks. In *IEEE Symposium on Computers and Communications*, pages 71–77, Alexandria, Egypt, June 1995.

[10] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *SOSP*, 2005.

[11] S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A tool for failure diagnosis in IP networks. In *MineNet Workshop at SIGCOMM*, 2005.

[12] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. IP fault localization via risk modeling. In *NSDI*, pages 57–70, 2005.

[13] S. Kullback. The Kullback-Leibler distance. *The American Statistician*, 41:340–341, 1987.

[14] M. Montemerlo et al. Junior: The Stanford entry in the Urban Challenge. *Journal of Field Robotics*, 25(9):569–597, 2008.

[15] A. J. Oliner, A. Aiken, and J. Stearley. Alert detection in system logs. In *ICDM*, December 2008.

[16] A. J. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *DSN*, 2007.

[17] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat. Pip: Detecting the unexpected in distributed systems. In *NSDI*, 2006.

[18] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat. WAP5: black-box performance debugging for wide-area systems. In *WWW*, 2006.

[19] I. Rish, M. Brodie, N. Odintsova, S. Ma, and G. Grabarnik. Real-time problem determination in distributed systems using active probing. In *NOMS*, 2004.

[20] H. A. Sturges. The choice of a class interval. *J. American Statistical Association*, 1926.

[21] S. Thrun and M. Montemerlo, et al. Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9):661–692, June 2006.