Fault Detection and Localization in Distributed Systems using Invariant Relationships

Abhishek B. Sharma, Haifeng Chen, Min Ding, Kenji Yoshihira and Guofei Jiang
NEC Labs America, Princeton, NJ
{absharma, haifeng, min, kenji, gfj}@nec-labs.com

Abstract—Recent advances in sensing and communication technologies enable us to collect round-the-clock monitoring data from a wide-array of distributed systems including data centers, manufacturing plants, transportation networks, automobiles, etc. Often this data is in the form of time series collected from multiple sensors (hardware as well as software based). Previously, we developed a time-invariant relationships based approach that uses Auto-Regressive models with eXogenous input (ARX) to model this data. A tool based on our approach has been effective for fault detection and capacity planning in distributed systems. In this paper, we first describe our experience in applying this tool in real-world settings. We also discuss the challenges in fault localization that we face when using our tool, and present two approaches – a spatial approach based on invariant graphs and a temporal approach based on expected broken invariant patterns – that we developed to address this problem.

I. INTRODUCTION

The state-of-the-art sensing and communication technologies allow us to collect massive amount of monitoring data from complex, distributed systems, if needed. For instance, we can deploy hardware and software sensors to collect monitoring data at multiple vantage points in today’s data centers including servers and network switches, power supply and cooling equipments, operating system (OS) and application performance metrics, etc. Similarly, physical plants deploy a sophisticated monitoring infrastructure [1], and modern automobiles have hundreds of sensors [6]. However, mere visibility is not enough. We need to extract information from monitoring data to manage complex, distributed systems more effectively. Big data analytics for system operational intelligence is an active area of research both within academia and industry. Innovative emerging products in this domain include solutions for IT [2] as well as physical systems [1].

Monitoring data from a variety of systems is in the form of time series, e.g. OS performance metrics from servers, network traffic measurements, environmental data on temperature, pressure, etc. for physical plants. Historically, such data has proved extremely useful; for instance, one can detect faults or anomalies in a system by comparing time series measurements from a sensor, i.e. a performance metric, against a threshold. There are several well-known parametric (e.g. auto-regressive models) [12] as well as non-parametric techniques (e.g. based on clustering) for modeling time series data; several of these have been adapted for detecting faults and anomalies [18], [5].

Over the past few years, we have built a product called SIAT for modeling time series monitoring data [13]. It uses Auto-Regressive models with eXogenous input (ARX) to model dependencies between different metrics ¹. We refer to dependencies between two or more metrics that hold across time as invariants. E.g. the instantaneous resource (CPU, memory, etc.) utilization at a web server will depend on the number of HTTP requests that it is serving and we can model such dependencies as invariants [13]. These invariant relationships can be used to monitor a dynamic system and detect faults and anomalies in it. We describe this approach using an example next.

Example (Part I): modeling using invariants. Figure 1(a) shows the graph of invariant ² relationships extracted from a real-world time series dataset³ consisting of 20 metrics. Each node in the graph represents a metric and a link between two nodes denotes an invariant relationship between them. This invariant graph has 39 edges, and 5 connected components. The largest connected component consists of 11 nodes, and consists of two densely connected sub-components with a common node. There are three isolated invariants, nodes with degree 1, and a small connected component with three nodes. Hence, we can build a model for normal system operation from time series measurement data using invariants, and represent it as a graph. The structure of this invariant graph (e.g. connected components, degree distribution, etc.) provides global information on dependencies across metrics.

Example (Part II): fault detection using invariants. If at some point the system measurement data does not “fit” some (or all) of the invariant relationships, i.e. invariants are broken, then we know that something has happened (possibly a fault or an anomaly). Figure 1(b) shows the broken invariants (dashed lines). ¹In this paper, we use the term metric to refer to both a sensor (or a resource such CPU, memory) and the time series monitoring data associated with it.
²We define an invariant later in Section II.
³We describe this dataset in more detail later in Section VII-A.

Fig. 1. (a) Invariant graph (b) Broken invariants (dashed lines).

Example (Part I): modeling using invariants. Figure 1(a) shows the graph of invariant ² relationships extracted from a real-world time series dataset³ consisting of 20 metrics. Each node in the graph represents a metric and a link between two nodes denotes an invariant relationship between them. This invariant graph has 39 edges, and 5 connected components. The largest connected component consists of 11 nodes, and consists of two densely connected sub-components with a common node. There are three isolated invariants, nodes with degree 1, and a small connected component with three nodes. Hence, we can build a model for normal system operation from time series measurement data using invariants, and represent it as a graph. The structure of this invariant graph (e.g. connected components, degree distribution, etc.) provides global information on dependencies across metrics.

Example (Part II): fault detection using invariants. If at some point the system measurement data does not “fit” some (or all) of the invariant relationships, i.e. invariants are broken, then we know that something has happened (possibly a fault or an anomaly). Figure 1(b) shows the broken invariants (dashed lines). ¹In this paper, we use the term metric to refer to both a sensor (or a resource such CPU, memory) and the time series monitoring data associated with it.
²We define an invariant later in Section II.
³We describe this dataset in more detail later in Section VII-A.
lines) at a certain time. We can see that all broken invariants have metric \(m_l\) in common, and hence, in this case, the component associated with \(m_l\) is likely to have a fault. Thus, using the pairwise or multivariate invariant relationships between metrics we can transform the task of anomaly or fault detection in distributed systems into a problem of identifying unusual nodes in a graph. This approach has been applied for fault and anomaly detection in several IT systems [7], [13].

**Our contributions.** This paper makes two contributions. First, we describe the key algorithmic ideas that are implemented in SIAT, and describe our experiences with applying SIAT to real-world problems (see Sections II-III). Dealing with real-world datasets exposed two challenges, namely improving the fault localization algorithm of SIAT, and reducing the second contribution is that we formulate these challenges as an input (ARX) [12]. An ARX model with parameters and due to measurement noise (see Section IV). Our world datasets exposed two challenges, namely improving the real-world problems (see Sections II-III). Dealing with real-world problems (see Sections II-III).

**Our contributions.** This paper makes two contributions. First, we describe the key algorithmic ideas that are implemented in SIAT, and describe our experiences with applying SIAT to real-world problems (see Sections II-III). Dealing with real-world datasets exposed two challenges, namely improving the fault localization algorithm of SIAT, and reducing the second contribution is that we formulate these challenges as an input (ARX) [12]. An ARX model with parameters and due to measurement noise (see Section IV). Our world datasets exposed two challenges, namely improving the real-world problems (see Sections II-III). Dealing with real-world problems (see Sections II-III).

**II. System invariants and SIAT**

In this section we first describe a framework for discovering pairwise relationships in a large time series dataset that we first proposed in [13]. We then highlight the key steps in invariants based fault/anomaly detection that we built into the SIAT.

**A. System Invariants**

An invariant is a pairwise relationship between two time-series expressed as an AutoRegressive model with eXogenous input (ARX) [12]. An ARX model with parameters \((n,m,k)\) between time series \(y(t)\) and \(x(t)\) is shown in (1).

\[
y(t)+a_1y(t-1)+a_2y(t-2)+...+a_ny(t-n) = b_0x(t-k)+b_1x(t-k-1)+...+b_mx(t-k-m) \tag{1}
\]

\(x(t)\) is the exogenous input time series and the parameters \(k\) and \(m\) determine the number of previous samples of \(x(t)\) affecting the current value \(y(t)\). The parameter \(n\) controls the extent of autocorrelation in \(y(t)\) – i.e. the current value \(y(t)\) depends on \(n\) previous samples. We denote the model’s coefficients as \(\theta\) where

\[
\theta = [a_1, ..., a_n, b_0, b_1, ..., b_m]^T \tag{2}
\]

We can rewrite (1) using vector notation as shown in (3) where we use to notation \(y(t|\theta)\) to denote the relationship between \(y(t)\) and \(\theta\).

\[
y(t|\theta) = \psi^T \theta \tag{3}
\]

where

\[
\psi = [-y(t-1), ..., -y(t-n), x(t-k), ..., x(t-k-m)]^T
\]

For a fixed \((n,m,k)\), we can estimate \(\theta\) using least squares regression. Assume that we have measured \(x(t)\) and \(y(t)\) over a time interval \(1 \leq t \leq N\). Using (3) we can define a system of linear equations and compute an estimate for \(\theta\) by minimizing the mean squared error; (4) defines the mean squared error.

\[
MSE(\theta) = \frac{1}{N} \sum_{t=1}^{N} (y(t) - \hat{y}(t|\theta))^2 \tag{4}
\]

where \(y(t)\) is the measured value at \(t\) and \(\hat{y}(t|\theta)\) is our estimate for it using the ARX model, i.e. (3).

The ARX model cannot capture all kinds of pair-wise relationships between two time series. For instance, if \(y(t)\) has a non-linear dependency on \(x(t)\) then the ARX model cannot capture this relationship. Given a pair of time series, how do we decide whether an ARX model captures their relationship?

We use a normalized fitness score \(F(\theta)\) as a measure of the “goodness of fit” of an ARX model [13]. \(F(\theta)\) is defined as follows:

\[
F(\theta) = 1 - \sqrt{\frac{\sum_{t=1}^{N} |y(t) - \hat{y}(t|\theta)|^2}{\sum_{t=1}^{N} |y(t) - \bar{y}|^2}} \tag{5}
\]

where \(\bar{y}\) is the mean of the observed values \(y(t)\). \(F(\theta)\) is always less than one and a high \(F(\theta)\) value indicates that the ARX model fits the observed data well. In practice, we specify a threshold \(\tau\) and if the fitness score of an ARX model for \(y(t)\) and \(x(t)\) is greater than \(\tau\), then we declare that there exists a time-invariant linear relationship, i.e. an invariant, between them; the ARX model captures this relationship.

**B. SIAT**

We built SIAT based on our idea of system invariants. It has proved effective at two important system management tasks, fault/anomaly detection and capacity planning [7], [13], [14]. The two main design decisions in building SIAT were (1) ARX model identification, i.e. what should we set \(n, m, k\) to and (2) an algorithm for fault detection?

A simple approach for ARX model identification is to consider all possible values for \(n, m, k\) within a range, and amongst the models with fitness score greater than \(\tau\), pick the one with the highest fitness score. For IT systems such as multi-tier web services, and Oracle and SQL databases, a choice of \(0 \leq n, m, k \leq 2\) has worked well [13], [7], [14], and for such a small range for model parameters SIAT’s brute force search is not prohibitively expensive. A more efficient approach for invariant model identification remains an open problem.

SIAT uses invariants for detecting faults and anomalies in real time as described next. At each time \(t\), it keeps track of the residual between the measurement \(y(t)\) and its estimate \(\hat{y}(t|\theta)\) defined as:

\[
R_{xy}(t) = |y(t) - \hat{y}(t|\theta)| \tag{6}
\]

In the absence of faults or anomalies, we would expect \(R_{xy}(t) \leq \varepsilon_{xy}\), where \(\varepsilon_{xy}\) is a threshold determined by the ARX modeling error. Hence, if \(R_{xy}(t) > \varepsilon_{xy}\), SIAT declares the invariant between \(x(t)\) and \(y(t)\) as broken. If the same invariant is broken for three consecutive samples, then SIAT raises an alarm and marks the corresponding metrics for
further investigation by the system administrators. We made this design choice to reduce the possibility of false alarms and it has worked well in practice. Each invariant has a different threshold $\varepsilon_{xy}$ associated with it. SIAT uses the residuals from the training data to automatically set $\varepsilon_{xy}$ as,

$$\varepsilon_{xy} = 1.1 \times \arg_{\varepsilon_{xy}} \left\{ \text{Prob}(R_{xy}(t) < \varepsilon_{xy}) = 0.995 \right\} \quad (7)$$

i.e., it chooses a value $r_{xy}$ that is greater than 99.5% of the residuals observed on the training data and sets $\varepsilon_{xy}$ to be 10% larger than $r_{xy}$.

Figure 2 shows a screenshot from SIAT. SIAT extracted an invariant ARX relationship between $y(t)$ and $x(t)$ from the training data (not shown in Figure 2), and the top plot shows the residual $R_{xy}(t)$ for the $x(t)$ and $y(t)$ measurements shown in the middle and bottom plot, respectively. Initially $R_{xy}(t)$ lies with its bounds$^4$ (yellow horizontal lines in top plot), but then it starts increasing, and converges at a value higher than SIAT’s upper bound for it. The corresponding measurements for $y(t)$ are marked out in the bottom plot. SIAT marks the invariant between $y(t)$ and $x(t)$ as broken for the duration when $R_{xy}(t)$ is outside its bound.

III. EXPERIENCES WITH SIAT

Success with IT systems. SIAT has been effective in tackling a range of problems in distributed, computing systems including fault and anomaly detection, generating signatures for recurring failures, and capacity planning [13], [14], [7], [8]. Figure 1 and the discussion on it in Section I outline how SIAT can be used for fault/anomaly detection. For recurring failures such as memory leaks, busy loops, DoS attacks on web services, etc., we can infer a pattern for invariants that break when a particular kind of fault occurs, and use it as a signature in future [8]. SIAT can also perform what-if analysis, often for capacity planning, using invariants. For example, an ARX model between the number of HTTP requests and CPU usage at a web server allows an administrator to estimate how much more CPU power is needed to maintain the current system performance if the rate of HTTP requests is doubled.

Model training time. We are constantly striving to improve the time it takes to extract system invariants from training data. Recall that we estimate an ARX model’s parameter using least-squares regression, and SIAT estimates $O(n^2)$ models for a dataset with $n$ metrics. For 24x7 monitoring data collected from multi-tiered infrastructure supporting web services, enterprise applications, and data warehouses, SIAT’s training phase duration is on the order of minutes. (We can test for broken estimates in real-time since computing $\hat{y}(t|\theta)$, given the ARX model parameters, only involves a small number of multiplications and additions.) With very large datasets, disk IO becomes the bottleneck, and we are currently working on parallelizing SIAT’s training phase using MapReduce-style approaches.

SIAT for physical systems. Recently, we applied SIAT to analyze time series measurement data collected from physical systems such as manufacturing plants and automobiles [9]. Since the monitoring data is in the form of time series, it can be analyzed using SIAT. However, SIAT’s assumption that $0 \leq n, m, k \leq 2$ does not always hold for physical system data. In particular, with high sampling frequency (we know of one monitoring system that can collect samples every 10 ms or less), the correlation between metrics $x$ and $y$ can be significant even for a large lag, i.e. $k >> 2$, and we may also need to estimate large models, i.e $m, n >> 2$. We are exploring more sophisticated techniques for ARX model identification that will scale better than our current brute force approach.

Quality of data raises additional challenges when analyzing data from physical systems. We have had to deal with many missing samples in the time series data, and samples corrupted by (Gaussian) measurement noise. We are currently augmenting SIAT with multiple modules that pre-process data to improve its quality before it is fed to SIAT.

IV. LESSONS LEARNT: REAL-WORLD CHALLENGES

While SIAT is extremely effective at detecting faults and anomalies, localizing these faults can be challenging. In one of our experiments with a multi-tier web hosting infrastructure consisting of a web, an application, and a database server, we collected data for 111 metrics [13]. We then injected a busy loop fault in the application server. Out of the 34 different application server metrics, the seven CPU related metrics (utilization, idle time, IO wait time, etc.) are impacted the most by this fault. Isolating these metrics is important for three reasons: (1) it saves debugging effort – e.g. in this instance administrators can focus on the application server first, and look at the broken invariants at web and database server after fixing the application server, if needed (2) it facilitates root cause analysis – e.g. a high CPU load without any significant change in the number of HTTP requests indicates rogue computation somewhere, and (3) it allows us to infer signatures for faults that repeat often [8].

We refer to the problem of identifying the most abnormal metrics as the metric ranking problem. Solving it involves separating true SIAT alarms from false ones. It is challenging to do this in the presence of measurement noise. When applying SIAT to real-world data, we often observe that a

$^4$The upper and lower bounds are automatically computed using $\varepsilon_{xy}$ (see eq. 7).
small fraction of invariants are broken even in the absence of any apparent faults or anomalies. Additionally, the set of metrics with such broken invariants changes over time. This is not entirely surprising; it is too simplistic to expect any parametric model to hold at all time points. Hence, to solve the metric ranking problem, we need to differentiate between invariants broken due to a fault or anomaly from ones affected by (measurement) noise. We refer to this second challenge as the problem of noise reduction. Next we describe two approaches for tackling these problems – one leverages the invariant graph structure (Section V) and the other is based on temporal patterns of broken invariants (Section VI). Preliminary evaluation results for these two algorithms are encouraging (see Section VII). Currently, we are doing more tests, and plan to add these algorithms to SIAT to improve its fault localization capabilities and robustness to measurement noise.

V. SPATIAL APPROACH: MINING THE INVARIANT GRAPH

Given a dataset of time series from multiple metrics, we can construct an invariant graph using the system invariants extracted by SIAT. The nodes correspond to the metrics and edges represent the pair-wise invariant relationships between these metrics. The invariant graph captures a spatial view of a system in terms of both intra and inter-component dependencies captured by invariants. During the operational or testing stage, at any given time \( t \), some of the invariants may be broken, and an invariant graph can capture this information as well; see Figure 1(b).

By differentiating edges in an invariant graph based on whether or not the corresponding invariants are broken, we can associate a score with each node in the graph. The main idea of our spatial approach is to define a score for nodes in the invariant graph that is directly proportional to the abnormality of their corresponding metrics. We can then use this score to filter out invariants broken due to noise as well as rank abnormal metrics.

A. Node scores

We define two scores for each node based on the invariant graph. nodeScore captures the information extracted from the edges incident at a node while neighborScore aggregates information across the one-hop neighbors of a node.

**Definition 1** nodeScore: Given an invariant graph \( G(V,E) \) at time \( t \) consisting of vertices in \( V \) and edges in \( E \), let \( d(v) \) denote the degree of a vertex \( v \) and \( be_i(v) \) denote the number of broken edges (invariants) incident at \( v \). The nodeScore of \( v \in V \) is defined as:

\[
nodeScore_t(v) = \frac{be_i(v)}{d(v)}
\]  

Note that the set of broken invariants associated with a metric can change across time causing \( be_i(v) \) to change as well, and hence the nodeScore of a vertex will change over time as well. \( d(v) \) is equal to the number of invariants we learn for the metric associated with node \( v \) and does not change over time. SIAT currently uses nodeScore to rank abnormal metrics.

However there are scenarios in which nodeScore is not sufficient. Consider the connected component of the invariant graph consisting of nodes \( A, B, \) and \( C \) shown in Figure 3. A solid line for an edge denotes that the corresponding invariant is still valid while a dashed line indicates a broken invariant. Both the edges incident on node \( A \) are for broken invariants, and hence \( nodeScore_t(A) = 1 \) while \( nodeScore_t(B) = nodeScore_t(C) = 0.5 \) because the edge \( e = (B,C) \) is not broken. Hence, we can declare node \( A \) to be the most abnormal but what about nodes \( B \) and \( C \)? The fact that the edge between nodes \( B \) and \( C \) is not broken and both these nodes have a broken edge with \( A \) is a strong indication that node \( A \) is the only abnormal node; an anomaly at either \( B \) or \( C \) would cause the edge \( e = (B,C) \) to be broken as well. This example underscores the value of looking at all the edges amongst a group of nodes together.

We capture this intuition by defining another score.

For an invariant graph \( G(V,E) \), we define the broken-invariant-neighboring-nodes of a node \( v \in V \) at time \( t \), \( BINN_t(v) \), to be the subgraph of \( G(V,E) \) consisting of the neighbors of \( v \) that are connected to it by a dashed edge (corresponding to a broken invariant) and all their incident edges. E.g., in Figure 3, \( BINN_t(G) \) is the subgraph with nodes \( \{F,I,J,K\} \) and edges \( (F,I), (I,J), \) and \( (J,K) \); \( L \) is not included because its invariant with \( G \) is not broken. Based on \( BINN_t(v) \), we define the neighborScore of a node \( v \).

**Definition 2** neighborScore: Given an invariant graph \( G(V,E) \) at time \( t \) that determines the \( BINN_t(v) \) for each node \( v \in V \), the neighborScore of \( v \) is defined as:

\[
neighborScore_t(v) = 1 - \frac{\# \text{ broken edges in } BINN_t(v)}{\text{Total } \# \text{ edges in } BINN_t(v)}
\]  

Based on Figure 3, \( neighborScore_t(G) = 1/3 \).

Special cases for neighborScore: When computing neighborScore a special case arises when \( BINN_t(v) \) does not contain any edges. For example, in Figure 3 nodes \( D \) and \( E \) have only one (broken) edge and hence the subgraphs \( BINN_t(D) \) and \( BINN_t(E) \) contain a single isolated node (\( E \) and \( D \), respectively). We refer to such invariants as isolated invariants. With isolated invariants, neighborScore does not have any discriminative power, i.e. looking at the one-hop neighbors of its nodes does not provide us with any information. Hence, we set the neighborScore to zero. This is a design choice. An alternative is to set the neighborScore to one, but with this choice both nodeScore and neighborScore for nodes \( D \) and \( E \) will be one. This will lead to their corresponding
metrics getting the highest rank amongst abnormal metrics. However, in several real-world cases, we have found that typical faults and anomalies are more likely to affect nodes in larger components of an invariant graph, e.g. the component containing $G$, than isolated invariants. Accordingly, we want to bias the $neighborScore$ against isolated invariants, and hence, our design choice.

$neighborScore$ is not useful in another scenario. If all the invariants corresponding to a completely connected component in the invariant graph are broken, then every node from this component will have $neighborScore = 1 - (1/1) = 0$. However, their $nodeScore$ will be 1 making it likely that they will still get a high overall score. We intend for the $neighborScore$ to supplement the discriminative power of $nodeScore$, and this motivates our metaranking approach described later.

**Other scores**. It is possible to design different versions of the $nodeScore$ and $neighborScore$. For example, we can define weighted $nodeScore$ and $neighborScore$ by associating weights with edges. The fitness score of the invariant associated with an edge is an obvious choice for its weight. SIAT supports weighted scores but they did not achieve better results in isolating faults on the real-world datasets that we analyzed.

### B. Metaranking

**How do we combine the two scores, $nodeScore$ and $neighborScore$, for fault localization?** We can rank metrics based on the (weighted) average of the two scores. Another option is to first create two separate rankings for nodes, one for each score, and then combine the two ranked lists. This idea is inspired by previous work on metasearch and information retrieval [10], [4] where the goal is to combine the ranked lists of documents (or web pages) retrieved by multiple search engines or information retrieval systems, in response to a query, to produce a final list. It is well known that combining the results from different retrieval algorithms often improves performance compared to using a single retrieval algorithm [4].

Lee provides a rationale for when one should perform evidence combination (i.e. combine multiple scores or ranked list of documents) as different runs (of a variety of representations of a query) might retrieve similar set of relevant documents but retrieve different sets of non-relevant documents [15]. In our context Lee’s rationale translates into the following intuitions for defining $nodeScore$ and $neighborScore$: (1) all of them assign a (high) positive score to abnormal nodes, and (2) the set of normal nodes incorrectly assigned a positive $nodeScore$ has small (ideally, no) overlap with the corresponding set for $neighborScore$; otherwise we will end up with false positives when identifying abnormal metrics. We combine $nodeScore$ and $neighborScore$ in two ways: (1) the spatial average algorithm uses their average as the final score for ranking, and (2) the spatial rank algorithm creates two separate ranked lists, one for each score, and then combines by first assigning a weight to each rank (highest rank gets the largest weight) and then summing up the weights for each metric to compute their final score. We evaluate these two algorithms in Section VII.

### VI. Temporal approach for noise reduction

Currently, SIAT tackles the *noise reduction* problem using a simple heuristic: an invariant is marked as broken only if it is broken for three consecutive samples. However, this approach is not robust when combined with high frequency sampling. For example, a transient measurement noise lasting for 100 ms can cause broken invariants for sampling intervals less than 30 ms but SIAT will ignore it if measurements are collected less frequently. We have encountered cases where operators dynamically adjust the sampling frequency for monitoring data. One power plant operator routinely collects samples 100 ms apart, but during critical phases of the plant’s operation, such as stress testing of the equipments, data is collected every 10 ms. Based on these experiences, we took a more principled approach to tackle the noise reduction problem. We try to infer a *temporal pattern* for the impact of abnormal samples of a time series (due to faults/anomalies) on an invariant relationship. The following example illustrates our new approach.

**Example** : Consider the invariant relationship shown in (10) between two time series from a real world dataset with residual threshold $\varepsilon_y = 0.91$. The parameters of the ARX model are $n = 2, k = 0$, and $m = 1$.

$$y(t) = -0.23(y(t-1) - 0.24y(t-2) + 0.32x(t) + 0.21x(t-1) - 2.04x(t-2)$$

We inject a fault in time series $y$ at time $t$ by increasing $y(t)$ by 20%, i.e. $\tilde{y}(t) = 1.2 \times y(t)$ where $\tilde{y}(t)$ denotes the abnormal measurement (other relevant data points $y(t-1)$, $y(t-2)$, $y(t+1)$, $y(t+2)$, $x(t-1)$, $x(t)$, $x(t+1)$, and $x(t+2)$ are the same as the original measurements). We observe residuals $R(t) = 27.7$, $R(t+1) = 6.4$, and $R(t+2) = 6.8$. Since these values are greater than $\varepsilon_y$, this invariant is broken. However, based on the *temporal pattern* of this invariant being broken for three consecutive samples, we can infer more information. With $n = 2$, we know that a fault affecting $y$ at time $t$ can result in the invariant being broken during the interval $[t, t+2]$; whether this actually happens also depends on the coefficients $a_0, \ldots, a_n$ and $\varepsilon_y$. Similarly, from (1), we can infer that a fault in $x$ at time $t$ can cause this invariant to be broken during $[t+k,t+k+m]$; with $k = 0$ and $m = 1$, this implies $[t, t+1]$. The two most parsimonious hypotheses that can explain the observed pattern are: (1) a fault affects $y$ at $t$, or (2) a fault affects $x$ at $t$ and $t+1$.

The above example introduces the notion of *temporal propagation* of the impact of a fault/anomaly on an invariant; i.e. a fault in $y$ at $t$ can cause broken invariants during the interval $[t, t+n] ([t+k, t+k+m])$. This property can be used to identify invariants broken due to noise. It can also improve fault localization by allowing us to infer an explanation for the observed pattern of broken invariants. For instance, in the above example, if the invariant was broken only at time $t$, then we could either ignore it or assign the metrics $y$ and $x$ a lower rank (compared to other abnormal metrics) because the broken invariant pattern differs from the expected one. However if
the invariant was broken only at $t$ and $t+1$ then this pattern matches the expected temporal behavior from a fault in $x$ at $t$ but does not match the expected pattern for a fault in $y$ at $t$. Hence, we can declare $x$ as the abnormal metric with high confidence; in contrast, SIAT’s current heuristic will miss this fault because the invariant is not broken for three consecutive samples.

**Algorithm 1** Localize faults in time and rank abnormal metrics

**Input:** set of all invariants $I$, list of broken invariants $L_t$ at each time point $t \in [T-w,T]$, current time $T$.

**Output:** an anomaly score for time $t$.

**for all** metric $m$ with broken invariants in $L_t$ do
  score$[m] = \text{metricScore}(m, \{L_j : j \in [t,T]\})$
**end for**

anomalyScore$[t] = \text{timeScore}(t, T, \{L_j : j \in [t,T]\})$

Raise alarms for time points $t$ with high anomaly score; at each of these time points focus on metrics with high scores for further investigation.

**Algorithm 2** timeScore: computes a score to support the hypothesis that an anomaly occurred at time $t$

**Input:** lists of broken invariants $\{L_j : j \in [t,T]\}$, set of all invariants $I$, current time $T$.

**Output:** an anomaly score for time $t$.

**for all** metric $m$ with broken invariants in $L_t$ do
  score$[m] = \text{metricScore}(m, \{L_j : j \in [t,T]\}, I)$
**end for**

anomalyScore$[t] = \sum_m \text{score}[m]$

**Algorithm 3** Assuming that anomalies happened at $t$, assigns a score to metric $m$ based on the match between expected temporal pattern of broken invariants and the observed pattern

**Input:** metric $m$, set of all invariants $I$, list of broken invariants $\{L_j : j \in [t,T]\}$.

**Output:** score for metric $m$.

initialize $\text{metricScore}[m] = 0$

initialize $\text{expectedMetricScore}[m] = 0$

for all $j \in [t,T]$ do
  for all invariant $i$ in $I$ with metric $m$ do
    if ARX implies $i$ be broken at $j$ then
      incr. $\text{expectedMetricScore}[m]$ by 1
    end if
    if $i$ is broken at $j$ then
      incr. $\text{metricScore}[m]$ by 1
    end if
  end for
end for

$\text{metricScore}[m] = \frac{\text{metricScore}[m]}{\text{expectedMetricScore}[m]}$

Algorithms 1, 2, and 3 define the main steps of our online temporal approach. Based on observations up to the current time $T$, we select an interval of length $w$, $[T-w,T]$, such that an anomaly at any $t \in [T-w,T]$ can cause a broken invariant at $T$. Using the ARX model we set $w = \text{arg max}(n, m+k)$. For each time $t \in [T-w,T]$, Algorithm 1 computes an anomaly score using Algorithm 2. Algorithm 2 computes this score assuming that all broken invariants at time $t$ are due to anomalies affecting their corresponding metrics. It calculates the anomaly score for $t$ as the sum of the score for each metric with broken invariants at time $t$. Algorithm 3 computes the score for a metric based on the match between the expected and the observed broken invariant pattern. Thus, the overall anomaly score for a time $t$ indicates support for the hypothesis that an anomaly occurred at this time while the score for a metric denotes how likely it is abnormal.

VII. EVALUATION AND A CASE STUDY

We evaluate our invariant graph based and temporal pattern based algorithms using real-world time series with injected anomalies. We select the anomaly types based on our experience working with data from multi-tier web hosting infrastructure, physical plants, and sensors in an automobile. Hence, our evaluation is based on anomalies observed in the wild but our methodology allows us to create instances that stress different aspects of our algorithms. We also discuss a real-world case study.

**Fault models.** The faults/anomalies that we observe most often in real datasets fall into three basic models: (1) **Spike**, a large positive or negative change in measured value at time $t$; often, this anomaly affects a single sample, (2) **Constant-shift**, the measurements from an interval are shifted by a constant, and (3) **Increased-variance**, the measurements from an interval appear to be “noisy” – i.e. have higher variance than the rest of the time series. We do not know the root causes for these types of faults/anomalies for our datasets. However, similar anomalies have been observed in other domains [18]. Due to page limits, we present results only for **spike** anomalies here; the results for the other two types were similar qualitatively.

**Data with injected anomalies.** We inject spike anomalies in a real-world dataset. It consists of one day of measurements collected by 1091 sensors monitoring a physical plant$^6$. We have 2880 measurements from each sensor. We extracted 324 invariants from among the 1091 metrics using SIAT. In order to have complete control on when these invariants will break, we inject anomalies into the same data that is used for extracting the invariants. Without these injected anomalies, naturally these invariants will hold for the entire dataset. Our choice of using the training data with injected anomalies for evaluation, instead of injecting faults in a separate dataset, was due to the lack of ground truth – with two other similar datasets we find broken invariants (even without injecting anomalies), and without ground truth we do not know if these are due to anomalies or measurement noise. Using this methodology, we can clearly explain why an invariant is broken, and hence, stress test various aspects of our algorithm. We discuss a real-world case study in Section VII-A.

We injected several instances of the three types of anomaly into multiple metrics. We first present the results for **noise free** case, i.e. only the injected anomalies cause broken invariants, and later for the anomalies with noise case.

**Anomalies without noise.** To inject spikes, we picked three metrics – $m_1$ with a high degree of 16 (i.e. lots of invariant relationships), $m_2$ with moderate degree of 7, and $m_3$ with only one invariant relationship. We randomly injected 10 spikes into each of these metrics. Table I shows the accuracy of $\text{nodeScore}$ alone and in combination with $\text{neighborScore}$, i.e. the two variants of the spatial approach – **spatial avg.** takes an average of the two scores and **spatial rank** combines their rankings. A

$^6$Due to non-disclosure agreement, we cannot provide any additional details on these sensors.
spike anomaly in $m_1$ at time $t$ leads to broken invariants during the interval $[t, t+4]$. Table I shows the number of time points (out of 4) for which metric $m_1$ is ranked the highest by each algorithm. We can see that all the three methods are able to localize the anomaly to metric $m_1$ for all the 10 instances. In two cases, anomaly at $t = 146$ and $t = 274$, $m_1$ does not get the top rank. This is because the metric $m_2$ is also affected by a spike anomaly then, and it is ranked first. We observed similar results for spike anomalies injected in $m_2$ and $m_3$. This is not surprising because our test case is simple and clean. However, it is still worth noting that the highest anomaly score does not occur at the time when the anomaly is injected; for $m_1$ there is a shift by 2 samples. Our temporal algorithm is able to localize the spike anomaly not only to the metric $m_1$ but also at the time point when it is injected (except for $t = 146$ and $t = 272$ cases where two metrics are affected; in these cases $t = 148$ and $t = 274$ are the points with the highest score). But it does this after observing all the broken invariants over a window of samples; e.g. in case of anomaly at $t = 146$, it looks at all the broken invariants during $[146, 149]$. This observation points to a trade-off between our spatial and temporal approaches in an online setting: the spatial approach can raise early alarms but it may not be able to pin-point the time at which a fault or anomaly occurred; the temporal approach can provide more accurate localization in time but with a delay.

Anomalies with noise. We mimic the measurement noise problem at a certain time $t$ by picking a few metrics at random and marking a certain fraction of their invariants as broken. We make sure that these metrics are different from the metrics with an injected anomaly. Thus, we have two categories of broken invariants – ones that are broken due to anomalies and others that are broken due to “noise”. We refer to the later group as noisy broken invariants.

To the injected spike anomaly case, we now add noisy broken invariants for three metrics. We consider two scenarios – 10% and 50% of a metric’s invariants broken due to noise.

Our metric for comparing different methods is their precision when their recall is 1. We compute these values as follows. Each method assigns a score to each metric with broken invariants (higher score indicates more abnormal), and ranks them in a descending order based on their score. In practice, operators go through the list of top-ranked metrics until they identify the abnormal metric. Hence, the ideal case for them is when the anomalous metric is ranked first. Consider the scenario where we have one abnormal metric and a method gives it the rank $k$. Then an operator has to look at the top $k$ metrics to localize the fault. Here, we say that the recall of the method is 1 and its precision is $1/k$.

Table II shows the average precision across 10 instances of spike anomalies in metric $m_1$ for different methods. We obtained qualitatively similar results for other metrics and omit the results due to page limits. nodeScore performs the worst because it looks at metrics in isolation and hence, cannot distinguish between invariants broken due to anomalies and noise. This situation worsens if noise has high impact – the precision for nodeScore drops to 0.46 and 0.4 for 10% and 50% noisy broken invariants. Contrast this against the noise-free case shown in Table I where nodeScore performs very well. The invariant graph based spatial algorithms, spatial avg. and spatial rank have reasonable precision even at 50% noisy broken invariants with spatial avg. outperforming spatial rank. This result is in agreement with Lee’s observation, in the context of document retrieval, that combining scores gives better performance than combining rank [15]. Our spatial algorithms are more robust to noise because they combine the “view” at a node (i.e. nodeScore) with the view across its one-hop neighborhood (i.e. neighborScore). This reduces the likelihood of metrics affected by (random) noise being ranked as abnormal.

Our temporal approach does not perform as well as the two spatial algorithms. The main reason for this is the incorrect localization of the anomaly in time. E.g. for a spike anomaly at $t = 272$, with noisy broken invariants, $t = 275$ has the highest anomalyScore (see Algorithm 2), and the precision of the temporal algorithm is 0.1. In contrast, the precision of the temporal algorithm if $t = 272$ is picked as the onset of anomaly is 1. (The results shown in Table II for the temporal approach are also averaged across all time points with broken invariants for a fair comparison with nodeScore and spatial algorithms.) Thus, these results expose an important challenge in using our temporal approach. If we know when an anomaly occurred, then the temporal approach is effective in identifying the abnormal metrics. For the noise-free case, the anomalyScore provides a good localization in time; it does not work as well with noise. We would like to point out that we do not need very precise localization in time for the temporal approach to be effective. For instance, the spike anomaly at $t = 272$ causes broken invariants during $[272, 275]$, and even with noisy broken invariants, the precision of the temporal algorithm, averaged over $[272, 274]$, is 0.75 (it drops to below 0.6 when the $t = 275$ is included). We plan to improve the robustness of our temporal algorithm as part of our future work.

<table>
<thead>
<tr>
<th>Anomaly@</th>
<th>max. anomaly score at $t$</th>
<th>nodeScore</th>
<th>spatial avg.</th>
<th>spatial rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>103</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>146</td>
<td>148</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>178</td>
<td>180</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>272</td>
<td>274</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>495</td>
<td>497</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1302</td>
<td>1304</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1409</td>
<td>1411</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1666</td>
<td>1668</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1822</td>
<td>1824</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2000</td>
<td>2002</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Table I

<table>
<thead>
<tr>
<th>Fault</th>
<th>Noise</th>
<th>nodeScore</th>
<th>spatial avg.</th>
<th>spatial rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>10%</td>
<td>0.46</td>
<td>0.9</td>
<td>0.74</td>
</tr>
<tr>
<td>$n_2$</td>
<td>50%</td>
<td>0.4</td>
<td>0.66</td>
<td>0.54</td>
</tr>
</tbody>
</table>

Table II

AVERAGE PRECISION OF DIFFERENT METHODS

TOTAL 4...


<table>
<thead>
<tr>
<th>Instance</th>
<th>Duration</th>
<th>maximum # broken inv.</th>
<th># metrics with broken invariants</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>19</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

TABLE III
CASE STUDY: INTERVALS WITH BROKEN INVARIANTS

A. Case study

We used our spatial and temporal fault localization algorithms to analyze a real-world data collected from 20 sensors that contains faults. The dataset is partitioned into training and test sets with each set consisting of one time series with 2800 samples for each sensor. Based on the information we received from the domain experts, the training set does not contain any anomalies but the test set does. However, we do not know which sensors capture these anomalies.

We extract 39 invariants from the training dataset. Figure 1(a) shows the invariant graph. For test data, we observe three intervals with multiple broken invariants. These intervals are shown in Table III along with the maximum number of broken invariants and the number of metrics associated with these broken invariants. For the instance shown in row 1 of Table III, we have 3 metrics, \( m_1, m_2, \) and \( m_3 \), with broken invariants. For all the 13 time points, their nodeScore is 0.28, 0.17, and 0.17, respectively \((m_1)\) has 2 broken invariants while \( m_2 \) and \( m_3 \) have one each). While none of the three metrics have high nodeScore, \( m_1 \) has a high neighborScore of 1 \((m_2 \) and \( m_3 \) have neighborScore of 0). Hence, the spatial algorithms – spatial avg. and spatial rank identify \( m_1 \) as the most abnormal metric. Further evidence for a fault in \( m_1 \) comes from the fact that for the samples in the other two instances (rows 2 and 3 in Table III), \( m_1 \) is consistently ranked the highest. This is despite new metrics (and their broken invariants) being present in later two instances. Figure 1(b) shows the broken invariants at one of the time points from row 2 in Table III. We manually inspected the samples from \( m_1 \) during these three time periods. The average value over these three windows is 7.2, 17.6, and 16.2 compared to 4.9 for the entire time series. Hence, this indicates a shift-by-constant type of fault. We verified with the customer that provided this dataset that SIAT’s detection, localization, and characterization of this fault was accurate.

VIII. RELATED WORK

We briefly discuss work on rank aggregation, and graph based anomaly detection that informed our two algorithms for fault localization using SIAT.

Rank aggregation. We mention work done in the field of information retrieval and metasearch for the web [10], [4], [15] that inspired our spatial algorithms in Section V-B. Global ranking problems also arise in other scenarios such as recommendation systems, voting, etc. Often, they involve computing a global rank using quantitative (e.g. a score between 1 and 5) or qualitative preferences from multiple users. With quantitative scores, we can use combination methods similar to our metaranking approach, and others used for information retrieval [15]. Ranking algorithms based on qualitative preferences [3] are not directly applicable to SIAT because it uses quantitative scores.

Graph based anomaly detection. Noble et al. propose a method for detecting anomalous subgraphs in a given graph [17] that can be used for detecting insider threats [11] and software bugs [16]. Our neighborScore is computed based on the subgraph consisting of the one-hop neighbors of a node but our goal is to identify abnormal nodes not subgraphs.

IX. CONCLUSION

We presented an overview of our tool SIAT for modeling dependencies in time series monitoring data collected from distributed systems, and described real-world problems that we have addressed using it. We also present two algorithms – one based on invariant graph and the other on temporal patterns – to solve the metric ranking and the noise reduction challenges that we encountered when analyzing real-world data using SIAT. A key contribution of this paper is to show that by using pairwise invariant relationships amongst time series monitoring data, we can transform the task of fault/anomaly localization in distributed systems into a metric ranking problem on invariant graphs in the presence of noise.

REFERENCES