Varanus: More-With-Less Fault Localization in Data Centers

Vaishali Sadaphal, Maitreya Natu, Harrick Vin
Tata Research Development and Design Centre,
Pune, India, 411013.
Email: {vaishali.sadaphal, maitreya.natu, harrick.vin}@tcs.com

Prashant Shenoy
University of Massachusetts,
Amherst, USA.
Email: shenoy@cs.umass.edu

***Abstract*—Detecting and localizing performance faults is crucial for operating large enterprise data centers. This problem is relatively straightforward to solve if each entity (applications, servers, business processes) within the data center can be instrumented and monitored explicitly. Unfortunately, such instrument-everything approach is often not tenable because of the limits imposed by enterprises on the permissible amounts of instrumentation intrusiveness and monitoring overhead. In this paper, we address the problem of achieving high accuracy of detecting and localizing performance faults in data centers, while minimizing the required instrumentation intrusiveness and overhead. We present novel algorithms for solving three key sub-problems: (1) How many monitors are required and where should they be placed within the data center? (2) Given the proposed instrumentation plan, how to detect the existence of performance faults accurately? and (3) How to localize the root-cause of the fault? We demonstrate the effectiveness of our approach for a real-world data center topology as well as through extensive simulations.**

I. Introduction

Today's enterprises run their mission-critical applications in data centers. Automatic detection of performance problems–resulting from software or hardware *faults*–and subsequent localization and correction of these faults is critical for the operation of most enterprises.

Fault localization of distributed data center applications is, however, a challenging task. Much of the difficulty in detecting and localizing faults stems from the inherent scale and complexity of enterprise data center applications. Consider, for instance, an equity trading application run by a top-tier investment bank in the US. This single application comprises 469 nodes/software components for processing incoming stock trades, 2,072 communication links between components, and 39,567 unique paths through which incoming stock trades flow in the system. Detecting performance problems (e.g., in end-to-end request processing latency) and pinpointing the problem to one or more culprit nodes is difficult in such complex systems.

Typically the effectiveness of fault localization depends on the number and the type of data collection monitors ("probes") available in the system. Retrofitting an operational system with the instrumentation required to facilitate fault localization, however, is always a challenge [1]. System operators and administrators are reluctant to introduce probes into the production environment, especially if the probes are intrusive (and can potentially modify the system behavior). Thus, a practical fault localization strategy should enable fault detection and localization with minimum instrumentation of the system. Much of the prior research in the area of fault localization [2], [3] has ignored this very basic practical requirement. This prior work has yielded sophisticated algorithms for fault localization, while assuming that all of the data required by the algorithm for its decision making can be easily gathered. Unfortunately, in most cases, collecting such data requires significant instrumentation of existing production data center systems, which makes it difficult to deploy in real-world operational systems.

In this paper, we advocate a "do more with less" approach for practical fault localization in large distributed data-center applications—our goal is to minimize the number of probes and reduce the amount of monitoring data necessary to detect and localize performance problems in the system. We implement our "do more with less" approach for detecting and localizing performance problems in a system called *Varanus*. In designing and implementing Varanus, this paper addresses three key questions.

1. *How many monitors are required and where should they be placed within the data center?* The instrumentation plan must achieve high-levels of fault detection and localization accuracy, while simultaneously minimizing the instrumentation intrusiveness and monitoring overhead. Varanus incorporates monitor/probe placement techniques to address this key challenge.

2. *Given the proposed partial instrumentation plan, how to detect the existence of performance faults accurately?* A monitor can detect a fault in the system by observing end-to-end performance of the on-going requests. With minimal instrumentation the problem of fault detection becomes complicated by the fact that multiple end-to-end paths may pass through each monitor. Thus, detecting node/path-level performance faults requires one to automatically detect significant changes in the composite time-series of latency values observed at each monitor.

3. *How to localize the fault to one or more culprit nodes?* A monitor can only detect the presence of fault at one or more of its upstream nodes. In order to correctly localize

978-1-4673-0298-2/12/$31.00

the fault to culprit nodes, the observation across multiple monitors needs to be consolidated to narrow down the fault to fewer nodes. Depending on the graph reachability properties, performance fault at a node, in theory, can be detected at one or more downstream monitors. However, various practical issues such as non-uniform use of paths, inaccurate fault detection, presence of multiple faults, etc. make the problem difficult.

In this paper, we present novel algorithms for addressing each of the above three questions, and thereby provide an *end-to-end solution* to the problem of detecting and localizing performance problems in complex distributed server applications. While Varanus can detect and localize performance problems within a distributed application, we presently do not focus on the problem of determining the root cause of the performance problem on the localized nodes; previously proposed techniques on root-cause determination [4] can be used for this purpose, in conjunction with Varanus.

As part of Varanus' design, we first present an algorithm that exploits the concept of *fault propagation* in graphs to instrument the smallest set of monitors for achieving the desired level of fault detection and localization accuracy. Second, we present a *self-tuning algorithm* for detecting significant changes in the latency distribution observed at each monitor; our algorithm uses the Student's t-test [5] as the basis for detecting significant changes in the observed latency distributions. The self-tuning allows the algorithm to automatically calibrate various parameters to minimize the false-positives and false-negatives in fault detection. Finally, we present a *probabilistic algorithm* that improves the accuracy of fault localization by combining the statistical confidence measure of fault detection at a monitor with the probability of observing a faulty node at the monitor.

We have evaluated the effectiveness of Varanus' algorithms using real-world application data and synthetic datasets. Our experiment results on a real-life topology show that we can achieve effective fault localization by deploying monitors only at 21% of the total nodes. The experiment results for sensitivity analysis of fault detection and localization show that Varanus can perform localization with greater than 80% accuracy with false positives and false negatives being less than 20%.

The rest of the paper is organized as follows. In Section II, we formulate the problem of detecting and localizing performance faults in data centers. We present novel solutions to the problems of monitor placement, performance fault detection, and root-cause localization in Sections III, IV, and V, respectively. We demonstrate the effectiveness of our approach in Section VI, and discuss related work in Section VII. We summarize our contributions in Section VIII.

II. Problem Formulation and Terminology

We model data centers supporting complex distributed applications and services as a *directed acyclic graph*, with nodes representing application processes and servers, and edges representing inter-process communication. Requests enter the system from any one of the source nodes, flow through a number of intermediate nodes, and exit from any one of the exit nodes.

In this paper, we focus on performance problems resulting from an increased end-to-end latency seen by a request from its entry at a source node to its exit at sink. An increased end-to-end latency is assumed to be caused by resource overload at one or more nodes. Our goal is to detect performance problems and identify the overloaded ("faulty") nodes contributing to these violations. A *monitor* is a node that is instrumented to measure the end-to-end latency incurred by each request from the time of its arrival into the system to the time when its processing is completed at the monitor. Each request carries with it the information of the time of its arrival into the system. The monitor observes the time at which the request departs from the monitor and thus computes the total time the request spent in the system thus far.

Our approach to fault detection and localization is based on the observation that node-level faults can be detected at downstream monitors by observing increase in the end-to-end latencies. Our approach involves three steps:
1. Identify and instrument a set of monitors so as to achieve high-levels of fault detection and localization accuracy, while minimizing the monitoring overhead (Section III).
2. Develop robust techniques for detecting increased end-to-end latencies (or fault), at each monitor and during each time interval (Section IV).
3. Finally, identify the potential root-cause node that resulted in increased end-to-end latency of requests (Section V).

The main insight behind our approach is that the effect of a faulty node is typically visible at multiple exit nodes. This is because; once a node is the cause of a performance fault, then requests passing through the node will experience degraded performance, and hence monitor nodes reachable from the faulty node will observe increased end-to-end request latencies. By identifying the monitor nodes that observe increased end-to-end request latencies, we can localize the fault to a unique node (or a small set of nodes). To formalize this notion, we define the notion of a node signature and an observed fault vector.

Node Signature: The signature of a node is the set of all monitors that are *reachable* from the node. Thus, given a set of k monitors, the signature S_i of a node i is a k-bit string where each bit represents the reachability of the monitor from the node. $S_{i,j}$ (the j^{th} bit of signature S_i) is set to 1 if monitor j is reachable from node i; otherwise, $S_{i,j} = 0$.

Fault Vector: Each monitor detects independently the possible occurrence of a fault by observing significant deviations in the end-to-end request latencies. Thus, for a data center instrumented with k monitors, a fault vector F is a k-bit vector where F_i (the i^{th} bit of fault vector F) is set to 1 when monitor i detects a possible fault; otherwise, $F_i = 0$.

As a result, our approach of localizing faults involves:
1. Identification of a set of k monitors such that every node has a unique signature and deriving a signature S_i $(\forall i \in [1, n])$ for each node in the graph with n nodes. It is desirable that we identify a minimal set of monitors so that the monitoring

overheads are minimized.
2. Construction of a k-bit fault vector F for each time-interval using the fault detection mechanism at the monitor nodes.
3. Matching of each node signature $S_i(\forall i \in [1, n])$ with the fault vector F to identify the potential root-cause node.

III. Monitor placement

Given a graph representing a distributed execution environment of a data center, the monitor placement problem can be defined as follows: Given graph $G = (V, E)$, find the set of monitors $M \subseteq V$ of *least cardinality* such that $\forall u \in V$, the signature of node u (denoted by S_u) is unique.

In the best case, the minimal number of monitors required to monitor n nodes will be $log(n)$. With $log(n)$ monitors n unique signatures can be constructed. In the best case, all nodes in V would be connected to the $log(n)$ monitors in such a way that n unique signatures can be constructed. A linear chain is another example topology, where $\frac{n}{2}$ nodes need to be chosen as monitors.

In what follows, we first describe two monitor selection algorithms – one based on deriving Hitting-Set (HS) and the other based on Information-Entropy (IE) measures of each node in V. While the Hitting-Set approach yields smaller number of monitors, the Information-Entropy based approach is more computationally efficient. We then present a hybrid approach that combines the best features of these two approaches. We observe that there is a trade-off between the required number of monitors and the uniqueness of node signatures. We address this trade-off by proposing an algorithm to reduce the number of monitors while bounding the maximum number of nodes with same signature.

While selecting monitors, we assume that monitors are perfect. If a node fails, a monitor placed at the node or any successor node will always detect that a failure has occurred upstream. We address the case of inaccurate detection in Section V. We generate signatures for single node failures and address the multi-node failure scenario in Section V. We assume a fixed topology and do not address scenario of dynamically changing graphs. Small changes in topology result in partially accurate signatures which can be addressed at the level of fault localization (Section V). In the event of significant topology changes, we propose to recompute the locations of monitors.

A. Hitting-Set (HS) algorithm

Let A be a set of elements, and let C be a set of subsets of elements of set A. Then the set H ($H \subseteq A$) is said to be a hitting-set of C if it contains at least one element from each subset in C. The hitting-set H of the least cardinality is said to be the minimal hitting-set for C.

We now reduce the monitor placement problem to the minimal hitting-set problem such that an optimal solution to the hitting-set problem can be used to derive an optimal solution to the monitor placement problem. Our reduction is based on the following intuition: For each pair of nodes in graph G, we construct a set of *differentiator nodes*; a node x is referred to as a differentiator node for the pair of nodes (a, b) if x is reachable from exactly one of the two nodes a and b, and not both. Formally, given a system graph $G(V, E)$, create a collection of differentiator sets C as follows: For each pair of vertices (V_i, V_j) where $V_i, V_j \in V$, construct a set $C_{i,j}$ as follows: [$C_{i,j}$ = $(Successor(V_i) \cup Successor(V_j))$ - $(Successor(V_i) \cap Successor(V_j))$] where $Successor(V_i)$ denotes node V_i and the set of nodes in G that are reachable from node V_i.

Note that the minimum hitting-set problem is NP-Complete. Given the set C consisting of the sets of differentiator nodes for each node pair in G, we use a greedy algorithm to compute an approximate solution for the minimum hitting-set problem for C. In particular, we select an element $V_i \in V$ that is present in maximum number of sets in C. We repeat this step until all elements in C are hit. The selected set of nodes represent the hitting-set H for C.

The hitting-set H thus derived precisely determines the set of nodes for placing monitors. Such a monitor placement strategy ensures that the signatures of all pairs of nodes in G are unique (and are different in at least one bit position). Given a set of n nodes, the complexity of this approach is $O(k * n^3)$ where k is the maximum number of monitors that can be placed.

We illustrate the derivation of a hitting-set using an example shown in Figure 1(a). For the purpose of this illustration, consider the problem of identifying a set of monitors that would yield unique signatures for node set $T = \{1, 2, 3, 4\}$. The algorithm first identifies the set of differentiator nodes for each pair of node T. For instance, for node pair $(1, 2)$, the set of differentiator nodes is $\{5, 6, 8, 10\}$, as these nodes are either reachable from node 1 or node 2 and not both. Figure 1(b) shows the differentiator set C for all pair of nodes in T. Then applying the greedy hitting-set algorithm to C yields $H = \{6, 7\}$. Placing a monitor each at nodes 6 and 7 yields a unique signature for each node in T (see Figure 1(b)). Note that the signature of node 1 is 00, which also represents a *no-fault* scenario. In order to distinguish such an *all-zero* signature from the *no-fault* scenario, we add one additional monitor at node 1, leading to unique 3-bit signatures for all nodes in T.

B. Information-Entropy (IE) Algorithm

The monitor placement solution using information entropy measures of nodes is based on the following intuition: Given a set of nodes V and a candidate monitor node m, the set of nodes V can be classified into two subsets based on their reachability to m. Nodes from which monitor node m is reachable form one set R, while the remaining nodes (from which monitor m can't be reached) form the other set U. This indicates that if a monitor is placed at node m, the signatures of nodes in R and U will differ at least at one bit (corresponding to monitor m). The process of splitting the sets R and U can be repeated similarly until every node in V can be assigned a unique signature. This results in a classification tree with intermediate nodes of the tree representing the monitors.

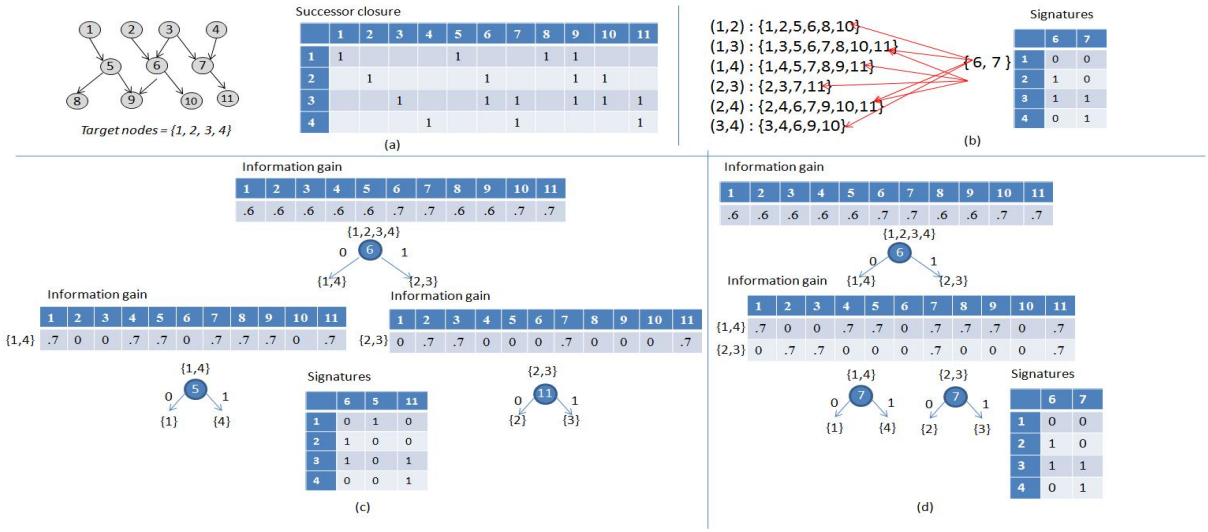

Fig. 1. Selection of monitors in an example topology shown in (a) using (b) Algorithm HS, (c) Algorithm IE, and (d) Algorithm IEHS.

Observe that the splitting of nodes into sets R and U for each candidate monitor node m impacts the height of the *classification tree*, and hence the total number of monitors required to create unique signature for all nodes in V. If a candidate monitor m splits the set of nodes into roughly equal size sets R and U, then this leads to a classification tree of low height and hence a smaller number of monitors. If, on the other hand, the resulting sets R and U have very different cardinality, then this can lead to highly unbalanced classification tree, and hence a larger number of monitors. Thus, to minimize the number of monitors, it is important to select the classification nodes (or monitors) that split the set of nodes into almost equal-sized sets R and U at each level of the classification tree.

To quantify the utility of a node in classifying a set of target nodes T at each level of the classification tree, we use the concept of information entropy as traditionally used in building decision trees. Formally, information entropy $H_m(T)$ for each candidate monitor node $m \in T$ is defined as:

$$H_m(T) = -p(T_m^R)log(p(T_m^R)) - p(T_m^U)log(p(T_m^U))$$

where $p(T_m^R)$ and $p(T_m^U)$, respectively, denote the fraction of nodes in T from which candidate monitor m is reachable or unreachable. Note that the information entropy of a node indicates the amount of randomness in the classification of the target set T. Nodes with low entropy split the set T into highly unequal sized R and U, while nodes with high entropy split T into roughly equal-sized R and U. Thus, to minimize the number of monitors, we select nodes with the highest entropy values at each classification stage. Ties are broken arbitrarily in the presence of multiple nodes with the same entropy value.

Figure 1(c) illustrates this information-entropy based approach with an example. Figure 1(c) shows the classification tree built for nodes in T. Observe that node 6 is reachable only from nodes $\{2,3\}$; hence, it splits the set T into two sets $U = \{1,4\}$ and $R = \{2,3\}$. The set $\{1,4\}$ is further classified using node 5, and the set $\{2,3\}$ is further classed using node 11. Thus, the set of monitors $\{5,6,11\}$ can define unique signatures for all nodes in $\{1,2,3,4\}$.

Note that, the hitting-set based approach explained earlier finds differentiator nodes for each node pair in G. The information-entropy based approach, on the other hand, recursively splits the entire set of nodes into smaller subsets to build a classification tree. The computational complexity of this algorithm is $O(k * n^2)$, where k is the maximum number of monitors to be placed in G.

C. The Information-Entropy Hitting-Set (IEHS) Algorithm

The hitting-set algorithm considers *all* node pairs and selects a monitor that can differentiate maximum number of nodes; thus, the algorithm makes a *globally-optimal* decision while selecting each monitor. This allows the hitting-set algorithm to identify close-to-optimal set of monitors. However, the need for considering all pairs of nodes makes the algorithm computationally-intensive. The information-entropy algorithm, on the other hand, makes a *locally-optimal* decision for each branch of the classification tree. This approach minimizes the height of the classification tree, but not the number of intermediate nodes in the classification tree (the number of intermediate nodes determines the number of monitors required). The *local* decisioning, however, makes the algorithm computationally more efficient.

In this section, we present a hybrid approach that is computationally more efficient than the hitting-set algorithm and achieves better results than the information-entropy algorithm. Like the information-entropy approach, we propose to incrementally build a classification tree by selecting monitors that provide best splitting of a given subset of target nodes. To assist this splitting, like the information-entropy approach, we compute the information entropy for each candidate monitor node. However, unlike the information-entropy approach, instead of making *local* split decisions for each subset of

nodes at a level in the classification tree, the hybrid approach considers the entire set of nodes at each level to find the best set of classification nodes. In particular, for each subset of target nodes $T_i, i \in [1, n]$ at a depth d, we build a candidate set of monitors M_i (with the largest information entropy value) that can classify T_i. Then, given this collection of subsets of the monitors $C = \{M_1, \ldots, M_n\}$, we select the monitor node that best hits all the sets in C. Unlike the traditional hitting-set approach, where the presence of a node in a set is considered a hit, we compute a weighted hit based on two factors: (1) the presence of a node in a set, and (2) the classification utility of a node in a set. This approach selects a set of monitor nodes that best classify *all* subsets of the nodes at each level of the classification tree. Given a set of n nodes, the complexity of this approach is $O(k * n^2)$, where k is the maximum number of monitors that can be placed.

The hybrid IEHS algorithm is illustrated in Figure 1(d). As before, node 6 is chosen as the root node of the classification tree forming two subsets $\{1, 4\}$ and $\{2, 3\}$. However, unlike the information-entropy algorithm, the IEHS algorithm builds the candidate set C at this level of the classification tree as: $C = \{(5, 8, 9, 7, 11), (7, 11)\}$, and then selects node 7 as the hitting-set for C.

Note that the proposed algorithm can also capture faults in the monitor nodes. This can be done by selecting monitor nodes such that each monitor node also forms a unique fault signature.

Trade-off between instrumentation and localization accuracy: In this section, so far, we have considered the problem of placing monitors that yield a *unique signature* for each node. One can reduce the number of monitors required by relaxing this unique signature requirement. In particular, we can define *cluster size* as the maximum number of nodes that can have the same signature, and then adapt the hybrid IEHS algorithm to identify a set of monitors that meet this relaxed requirement as follows: Split a set of nodes into subsets only if the cardinality of the set is greater than the cluster size. This simple adaptation enables us to trade-off the amount of instrumentation with fault detection and localization accuracy. In Section VI, we demonstrate that for a real-world topology, relaxing the unique signature requirement allows us to reduce number of monitors from 48% to 21% of the total number of nodes.

IV. Fault detection

Given a partial deployment of monitors the objective of fault detection is to detect the presence of faults by observing end-to-end latencies at the monitors. The requirement of limited instrumentation and intrusiveness presents several challenges in fault detection.

Limited instrumentation prevents per-node analysis: Fault detection can be fairly straight-forward if each node is instrumented to monitor its own processing delay. Presence of fault can then be detected simply by observing an increase in the processing delay of a node. However, in a scenario with the deployment of a limited number of monitors, the presence of fault in one or more nodes needs to be detected by observing the end-to-end latency of the requests arriving at the monitors.

Limited intrusiveness prevents per-request analysis: In the past, approaches have been proposed to track individual requests flowing through the system to classify requests based on the path and type [2], [3]. However, tracking of individual requests requires modification of applications. Such intrusive approaches are not found viable for deployment in many production systems.

In such a setup, the monitor observes a composite time series of the latency of various request types. The problem of fault detection is defined as:
Given a composite time series of end-to-end latencies consisting of multiple distributions (referring to multiple request types), detect the presence of significant and persistent change in one or more distributions.

In this paper, we propose to use statistical significance test to detect changes in such composite time-series. We propose to use Student's t-test [5] for detecting changes because Student's t-test works effectively even if the normality assumption of the data is violated. This is important since the request latencies may not always show normal distribution.

A. Proposed approach for fault detection

We divide the given time series of latency values into observation windows of size *window_size*. We compare the values in each observation window with that of the current behavior using the Student's t-test. Student t-test computes the probability of similarity (*p-value*) of distributions of the two sets of values. If the calculated *p-value* is less than a threshold (defined as *similarity_threshold*) chosen for the statistical significance, then it is concluded that the statistical properties of the two sets do differ significantly. In order to detect persistent changes, we define a variable *PersistenceFactor*. We detect a fault only if the change persists for more than *PersistenceFactor* number of windows. Fault detection can be performed in near-real-time after analysis of every temporal window.

The effectiveness of this approach depends on the appropriate tuning of the parameters viz. *similarity_threshold* and *window_size*. Incorrect value of *similarity_threshold* can result in detection of too many or too few changes. Incorrect value of *window_size* can result in incorrect capture of normal behavior of individual distributions. We next present techniques to automatically tune the algorithm parameters based on the input data properties.

Tuning of similarity_threshold: The expected amount of similarity between two windows showing normal behavior is different in different time series. The *p-value* considered to be normal in one data set is an indication of change in another data set. To compute the *similarity_threshold*, we first compute a *p-value* by running *t-test* on observation windows of the normal behavior data. The *p-value* thus obtained represents the expected amount of similarity between two current observation windows. We use this *p-value* as the *similarity_threshold*. The

similarity_threshold can be further tuned based on the user input of the *ChangeIntensity* parameter. The *similarity_threshold* can be set to a fraction *ChangeIntensity* of the above computed *p-value*.

Tuning of window_size: Given a composite time series consisting of multiple distributions, the algorithm demands that the size of the window, *window_size*, should be set such that sufficient data points corresponding to each distribution should be present in the window.

In order to set an appropriate observation window, we estimate the number of distributions present in the current observation window using Expectation Maximization Algorithm. The algorithm estimates the number of distributions, n, and the parameters (μ_i, σ_i, w_i) of each distribution i where μ_i, σ_i, and w_i are mean, standard deviation, and weight of each distribution. The weight of a distribution is the fraction of total points in the observation window that belong to the distribution. Let $w_{min} = min(w_i) \forall_i \in (1 \ldots n)$ be the minimum of the weights of all the requests. We then define the size of the observation window as, $window\_size = (\textit{MinDataPoints}/w_{min})$ where *MinDataPoints* is the minimum number of data points of a request type that should be present in the observation window so that an increase in its latency values is detected by the proposed algorithm. A value of *MinDataPoints* $>= 1000$ suffices to capture the data properties of a request type. This value has been computed empirically and has been observed to give accurate results.

The resulting self-adaptive fault detection algorithm (Algorithm SAFD) is run on all monitors to detect changes in the end-to-end latencies. When the fault detection algorithm on a monitor flags a fault, then the bit corresponding to the particular monitor in the *fault vector* is set to 1. The *fault vector* thus constructed by consolidating the observations of all monitors is further diagnosed by the fault localization algorithm presented in the next section.

V. Fault localization

Given an observed *fault vector* and the precomputed *signatures* of all nodes, the objective of fault localization is to identify the faulty nodes that best explain the observed fault vector.

A. Binary Signature Matching Algorithm (BSM)

A straight-forward approach is to match the observed fault vector with all node signatures to find the best match. The best matching node is declared as the likely root-cause. In the event where an exact match is not observed, we propose to compute string-edit distance between a *k*-bit binary signature S and a *k*-bit binary fault vector F. String edit distance here simply is the count of bits in F that are different from their corresponding bits in S. The nodes whose signature has the minimum distance from the observed fault vector are declared as the likely root causes.

B. Weighted Signature Matching Algorithm (WSM)

The accuracy of binary signature matching relies on (1) the accuracy of 1s and 0s in the observed fault vector and (2) the probability of observing the node signature as the observed fault vector. In practice, various factors such as nonuniform traffic, overlapping latency distributions, etc. might produce incomplete and inaccurate fault vectors. We address this issue in the following manner:

Compute weights for the bits in observed fault vector: First, we assign weights to the 1s present in the observed fault vector based on the confidence of the fault detection algorithm in observing a change. We compute this weight by computing the amount of change and probability of dissimilarity detected by the fault detection algorithm at a monitor.

Compute weights for the bits in node signature: Secondly, we assign weights to the 1s present in the node signatures by computing the probability of observing the fault in a node at a monitor. We compute the weighted node signature as follows: We propose to monitor the network traffic at the outgoing edge of each node. Note that we do not track the entire path followed by the request. We only monitor the outgoing traffic of each node. Based on the link weights, we compute the fraction of total number of requests passing through a node that are likely to pass through a monitor. We use this fraction to construct a weighted signature of a node. The weights thus reflect the amount of change observed by the fault detection algorithm.

Compute distance between the observed weighted fault vector and weighted node signatures: Given a weighted signature S of a node and weighted fault vector F each of length k, we compute their distance by comparing each value S_i in S with the corresponding value F_i in F as follows:

1) If $S_i = 0$ and $F_i = 0$ then Distance = 0
2) If $S_i > 0$ and $F_i > 0$ then Distance = 0
3) If $S_i = 0$ and $F_i > 0$ then Distance = F_i
4) If $S_i > 0$ and $F_i = 0$ then Distance = S_i

The key idea behind this approach is that if there is a mismatch between the bit values then the distance is proportional to the confidence in the observed 1 bit. We add the distance of each of the k values to compute the total distance between S and F.

Identify the likely root-causes: The node signatures with minimum distance from the observed fault vector are declared as the likely root causes.

We show through experimental results that Algorithm WSM performs better than Algorithm BSM in terms of accuracy. However, under reasonable non-uniformity levels of traffic, Algorithm BSM also works with high accuracy.

C. Addressing multiple faults (Composite Signature Matching)

In case of multiple faults, the fault vector observed at the monitors represents a composite effect of more than one faulty nodes. We make the assumption that the maximum number of simultaneous node faults at any point in time in the system is f_{max}. Given the fact that monitors are placed to determine unique signatures for single node fault, the scenario of multiple faults can be addressed in following two ways.

Single Signature Matching (SSM): Compute the string edit distance between the observed fault vector with signature

of all nodes and report the nodes with top f_{max} closest match.

Composite Signature Matching (CSM): Another approach is to pre-compute all combinations of k node faults ranging k from 1 to f_{max} and generate composite signatures for these combinations. Match the observed fault vector with the pre-computed composite signatures and determine the best matching combination(s).

The Single Signature Matching algorithm is computationally less intensive than the other approach but can miss out some of the faults in scenarios where some other non-faulty node forms a better match to the observed fault vector. The second approach on the other hand is computationally more intensive as it involves building composite signatures of all combinations. However, the second approach is less likely to miss out the actual faults. We later show through experimental evaluation that these algorithms effectively localize faults in a variety of multiple fault scenarios.

VI. Experimental evaluation

We present a real-world experiment to demonstrate the feasibility and effectiveness of deploying the proposed algorithms in a real-world setup. Through sensitivity analysis, we evaluate the accuracy of algorithms across a variety of network topologies and traffic patterns using simulation.

A. Application on a real-world case-study

We ran the monitor placement algorithm on a real-world data center topology which consists of 469 nodes. The monitor placement algorithm IEHS identified 225 nodes (48% of the total nodes) where the monitors need to be placed in order to obtain unique signatures for all the nodes. By increasing the maximum cluster size from 1 to 9, the required number of monitors decreased from 48% to 21%. Figure 2(a) presents the effect of increasing cluster size on the required number of monitors.

Most nodes can still be localized with a fine granularity even with monitors deployed at 21% of the nodes: An interesting observation is that, beyond a point, the increase in cluster size does not result in any significant decrease in the number of monitors (Figure 2(a)). A large increase in maximum cluster size (> 9) is ineffective because most of nodes in the topology form small size clusters. Figure 2(b) presents the distribution of number of nodes across different cluster sizes. Consider the scenario with maximum cluster size = 9. It can be seen that, even with a maximum cluster size set to 9, most of the nodes still form small-size clusters (size 1, 2, and 3). Thus, it can be inferred that even after decreasing the number of monitors from 48% to 21% by allowing a maximum cluster size of 9, most of the nodes can still be diagnosed with a fine granularity.

This behavior is explained by the structure of the network topology. A set of nodes can be part of a same-signature cluster only if these nodes contain the same set of monitors in their successor graphs. In other words, the successor graphs of these nodes overlap. Analyzing the network topology, we observed that most of the successor graphs tend to overlap only within small groups of 3 to 4 nodes. In Figure 2(c) we plot the group size of the nodes with overlapping successor graphs and the percentage of total number of nodes that belong to these group sizes. It can be seen from Figure 2(c) that 35% of the nodes belong to group size of 1 indicating that successor graph of these nodes are not overlapping with any other node. Furthermore, more than 80% of nodes tend to form such groups of size less than 9 nodes.

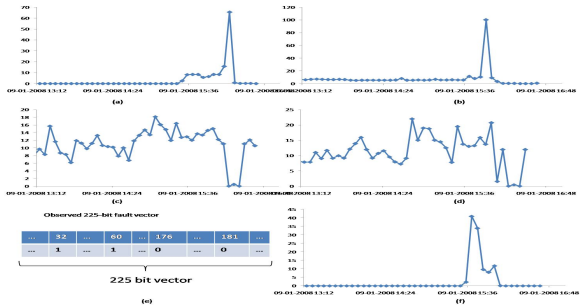

Fig. 3. (a), (b) End-to-end latency time-series observed at monitors 32, 60 respectively, reflects significant changes of node 227 resulting in a '1' bit in the fault vector, (c), (d) End-to-end latency time-series observed at monitors 181, 176 does not reflect changes of node 227 resulting in a '0' bit in the fault vector, (e) Observed 225-bit fault vector, (f) Node latency time-series for node 227 showing fault as increased latency,

IEHS outperforms other algorithms. It can be seen from Figure 2(d) that the Random selection algorithm selects very large number of monitors in comparison with other algorithms. The number of monitors selected by the IEHS algorithm is smaller than IE and HS algorithms.

After placing monitors on the chosen 225 nodes we ran the fault detection algorithm on these monitors. At a particular time some monitors observed an increase in the end-to-end latency (Figure 3 (a, b)) while many monitors did not observe any such increase (Figure 3 (c, d)). This behavior is captured by the fault detection algorithm and a 225-bit fault vector is computed (Figure 3 (e)). Based on signature matching, node 227 is identified as the likely root-cause behind the observed latency increase (Figure 3 (f)). Node 227 shows an increase in node latency in the similar temporal region as the signature nodes and thus is likely to cause the increase in the end-to-end latencies observed at the monitors.

B. Sensitivity Analysis

1) Setup: In order to evaluate the proposed algorithms over a wide range of networks we simulate request paths as DAGs (directed acyclic graphs) and generate different types of DAGs using [6]. We apply the monitor placement algorithm on these graphs to compute node signatures. We simulate workload using CSIM [7]. We inject faults at randomly chosen nodes by increasing the processing delay and run fault detection at the monitors.

We use the following terms to refer to the properties of the DAGs. The *size* of a graph refers to the number of nodes in the graph. The *fatness* of a graph represents amount of parallelism in the DAG. Value of fatness ranges from 0 to 1. A small

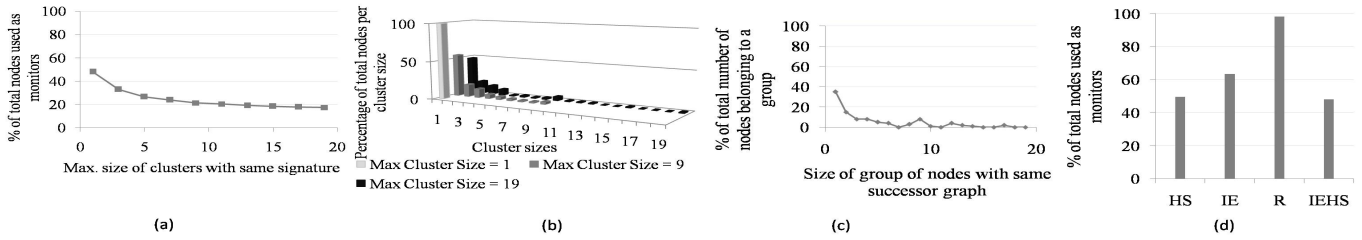

Fig. 2. (a) Effect of increasing cluster size on the number of monitors, (b) Distribution of the number of nodes forming different cluster sizes, (c) Distribution of size of groups with same successor graph, (d) Comparison of IEHS algorithm with HS and IE algorithm.

fatness value leads to a thin DAG (e.g., zero fatness results in a chain) with a low path parallelism, while a large *fatness* value induces a fat DAG with a high degree of parallelism. In the extreme case of *fatness = 1* almost all nodes in the topology are disconnected resulting in maximum degree of parallelism. In the following experiments we set the default value of fatness to 0.5. The *density* of a DAG determines the numbers of links in the DAG. Value of density ranges from 0 to 1, 0 indicating a very sparse graph and 1 indicating a very dense graph. In the following experiments we set the default value of density to 0.5.

2) Monitor placement: **Effect of increase in the number of nodes:** We generate network topologies with increasing network size and keep other properties of the graph such as *fatness* to a constant value. The monitor placement algorithms select monitors such that all nodes have unique signatures (cluster size = 1).

Network size does not affect the percentage of nodes used as monitors: Algorithm HS and IEHS compute smallest number of monitors and the percentage of monitor nodes remains constant for increasing network sizes (close to 50% and 45% respectively) (Figure 4(a)). Thus, the size of the network does not affect the percentage of total number of nodes selected as monitors. Instead, it is affected by other properties that define the structure of the graph, particularly *fatness* of the graph (discussed later).

In case of Random algorithm and IE algorithm, the number of monitors increases slightly with increase in the network size. With larger number of nodes in the network, the number of inappropriate choices for monitor placement increases. Random algorithm makes a random choice and the Algorithm IE makes a locally optimal choice. Hence, both these algorithms give less optimal results in presence of larger number of inappropriate choices and result in an increase in the percentage of total nodes used as monitors. Algorithm HS and Algorithm IEHS, on the other hand, make the most suitable choice at each level and thus are not affected by larger number of inappropriate monitor node choices.

Effect of increase in the *fatness* of the network As explained above, the *fatness* of a graph represents amount of parallelism in the directed acyclic graph (DAG).

Fatness affects the number of required monitors: Figure

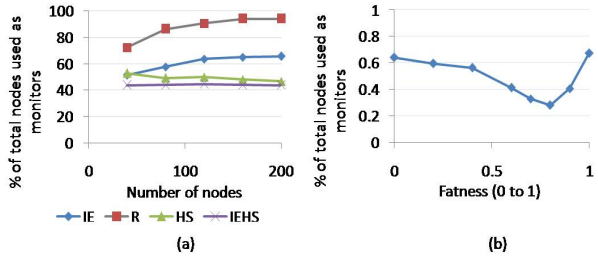

Fig. 4. (a) Effect of increasing number of nodes on required number of monitors (fatness factor = 0.8). (b) Effect of increasing fatness of graph on required number of monitors.

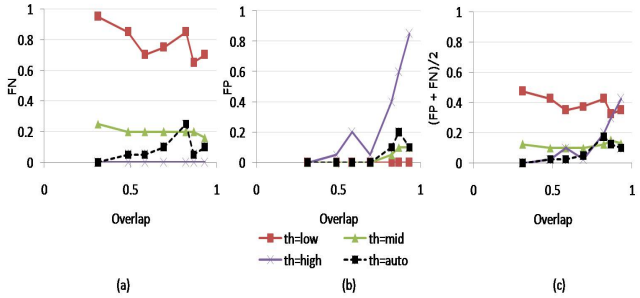

Fig. 5. Effect of auto tuning of threshold on false positives and false negatives.

4(b) shows the number of monitors computed by the IEHS algorithm for different fatness of a 100-node network. It can be seen that the required number of monitors decreases initially with increasing fatness in the graph. However, after a certain threshold of fatness, the number of monitors increase.

A DAG with fatness zero forms a linear chain like topology. In order to obtain unique signatures a chain topology requires every alternate node to be chosen as a monitor node. Thus an optimal algorithm would choose 50% nodes as monitors. The proposed algorithm being approximate in nature chooses approximately 60% of nodes as monitors.

With increasing fatness the number of parallel paths increases allowing fewer number of monitors to provide unique signatures. However, very large fatness values (close to 1) result in highly disconnected single-node graphs. Thus very fat graphs demand a large number of monitors to cover each of

the disconnected sub-graphs. The knee of this curve depends on the density of the DAG. The current experiments were performed on network with density = 0.5. Denser graphs (density close to 1 i.e. larger number of edges) will result in the knee point at a larger fatness value.

3) Fault detection: We evaluate the fault detection (Algorithm SAFD) on the basis of the generated false positives (FPs) and false negatives (FNs). False positive refers to no-fault events when the fault detection algorithm incorrectly detects a fault. False negative refers to the cases when the algorithm fails to detect the presence of fault.

Effect of overlap and auto-tuning of *similarity_threshold*: Recall that a monitor receives time series consisting of multiple distributions. The amount of overlap in the distributions affects the fault detection accuracy. We define overlap as the percentage of total number of values received at the monitor that can belong to more than one distribution. The effect of overlap can be addressed by appropriate tuning of the *similarity_threshold* value. As shown in Figure 5(a) and Figure 5(b), a small value of threshold detects only very large changes and thereby reduces false positives. However, small value of threshold (th=low) misses some actual faults thereby increasing false negatives. A reverse effect is observed with high threshold values (th=high) resulting in high false positives but low false negatives.

Auto-tuning of similarity_threshold is effective. It can be seen that the auto-tuned threshold (th=auto) results in both low false positives and low false negatives. Figure 5(c) shows that algorithm with auto-tuned threshold performs better than any of the static threshold settings.

Effect of skew and auto-tuning of *window_size*: The accuracy of the fault detection algorithm is also dependent upon the skew in the weights of distributions. High skew refers to cases where the number of data points of the distributions received within a window are very different and the observation window might not capture enough points of all distributions leading to incorrect inference of normal behavior. Such cases can lead to high false positives or high false negatives.

Auto-tuning of window_size is effective. Effect of skew can be addressed by setting appropriate window size. In case of high skew in the distribution weights, a large window size should be used such that enough data points of all distributions are captured in the window. Inaccuracy tends to decrease initially (window size = 4000 to 8000) with increasing window size because with larger window size the normal behavior is better captured (Figure 6(a)). Beyond a point (window size $>=$ 14000) the inaccuracy tends to increase because the system incorrectly considers even the change as part of normal behavior. The window size beyond which the inaccuracy tends to increase again depends on the location where the change starts taking place. In the experiments, we introduce a change after approximately 12000 data points.

Hence, the window size has to be large enough to capture data-points of all distributions, but should be small enough to avoid incorrectly capturing change as part of normal behavior. Figure 6(a) also shows that auto-tuning of window size correctly computes window size such that inaccuracy is low.

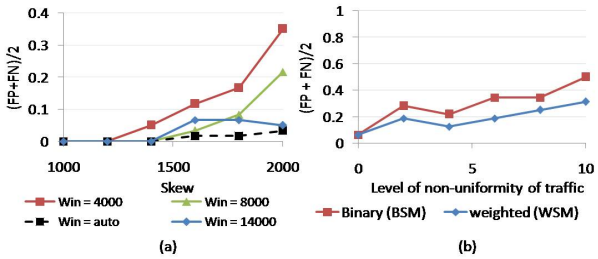

Fig. 6. (a) Effect of auto tuning of window size on false positives and false negatives. (b) Effectiveness of Weighted Signature Match (WSM) and Binary Signature Match (BSM) with increasing non-uniformity in traffic.

4) Fault localization: We evaluate fault localization on the basis of false positives and false negatives generated. The false positives measure the number of nodes incorrectly localized as the root-cause. The false negatives measure the number of actual faults that the algorithm fails to localize as the root-causes. We evaluate the effectiveness of fault localization in case of inaccurate fault detection and multiple simultaneous faults.

Effect of partial signatures: We evaluate fault localization algorithms (Binary Signature Match (BSM) and Weighted Signature Match (WSM)) by performing experiments with increasing non-uniformity of the traffic. The uniformity of traffic refers to the pattern of traffic being sent by a node on all of its outgoing links. A highly uniform traffic will send equal amount of traffic across all of its outgoing links. On the other hand, a highly non-uniform traffic will send traffic across its outgoing links in a highly skewed manner, sending very high traffic over a few links and very low traffic over few. As discussed earlier, the level of uniformity of traffic affects the probability of observing a node's failure at the reachable downstream monitors.

WSM outperforms BSM. As shown in Figure 6(b), with increasing non-uniformity the inaccuracy of Binary Signature Match (BSM) increases. The Weighted Signature Match algorithm (WSM) effectively captures non-uniformity and shows lower false positives and false negatives. For a non-uniformity level less than 6, Algorithm WSM demonstrates more than 80% accuracy in fault localization. The weighted signature matching (Algorithm WSM) captures the probability of traffic reaching the monitor node and the confidence in the change observed by the fault detection and hence results into more accurate signature match.

Effect of multiple faults: We next change the number of simultaneous faults occurring in the system. With larger number of faults the resulting fault vector is composite in nature and comprises of more than one node signatures. With this experiment, we test the performance of the algorithms SSM (Single Signature Match) and CSM (Composite Signature Match) in analyzing a *composite* fault vector.

CSM produces low FNs but high FPs. With increasing number of faults, CSM maintains near-zero false negatives while the SSM algorithm on the other hand tends to generate

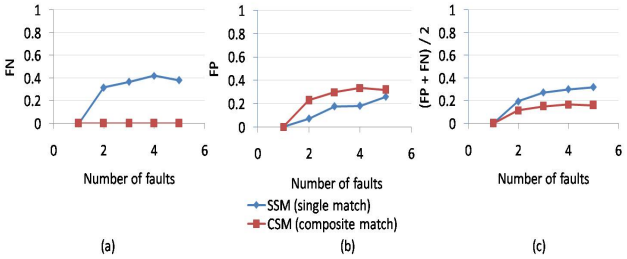

Fig. 7. Effectiveness of Composite Signature Match (CSM) and Single Signature Match (SSM) with increasing number of faults.

higher false negatives (Figure 7) . The composite signature of the nodes that have actually failed will always exactly match the observed fault vector (assuming high accuracy of fault detection). Hence the composite signature match results in near-zero false negatives. Single signature match is based on string-edit distance and signature of actual failed nodes might not always be the closest match to the observed fault vector and hence results in higher false negatives.

However, with increasing number of faults, the number of false positives generated by CSM are larger than the false positives generated by SSM. An incorrect match of a composite signature is more expensive than that of a single signature match. One incorrect match in case of single signature match will result in one false positive but one incorrect match in composite signature match will result in a set of nodes as false positives. The performance of both the algorithms could be further improved by using weighted signature matching.

VII. Related work

The problem of fault localization and performance debugging has been addressed by various researchers in the past.

Dependency model: Building dependency models [2], [3] requires knowledge of request flows which is not viable in many production systems. We simply use the per-hop network connectivity of the components for analysis. Techniques such as [8], [9] are also based on component-component dependencies. However, these techniques do not address the problem of monitor placement and fault detection. Our approach instead presents algorithms for systematic selection of monitors and fault detection. The model proposed by [8] can be leveraged by our approach for a finer localization. **Machine learning and statistical analysis:** Bayesian networks and statistical correlation techniques used in [4], [10] demand large amount of instrumentation. These can be used in conjunction with our approach for finer analysis. **Rule engine:** These systems perform diagnosis based on a set of pre-programmed rules [11] but are not generic in nature.

We next present related work in the context of the individual problems of monitor placement, fault detection, and fault localization. **Monitor placement:** Most of the proposed techniques in the past [2], [3], [4], [9] monitor all nodes to collect component-level metrics. Techniques proposed in [8] use limited placement of monitors but do not addresses the problem of systematic selection of monitors. **Fault detection:** The problem of fault detection has been typically addressed in three ways: reading system alerts [2], detection of SLO violations [4], and change detection [9]. The change detection techniques used in the past are applied on time-series consisting of a single distribution, such as latency time series of specific request or CPU utilization of a server, etc. However, because of the limited placement of monitors, the fault detection technique proposed in this paper processes a composite time-series consisting of multiple distributions. The proposed algorithm also automatically tunes various parameters based on data properties. **Fault localization:** Past techniques are based on rule chaining [11], belief propagation [8], [4], and signature matching [2]. The signature matching algorithms proposed in the past use binary signatures and fail to capture inaccuracies in signatures. We propose a weighted signature matching algorithm that addresses issues of multiple faults and signature inaccuracies.

VIII. Conclusion

We address the problem of achieving high accuracy of detecting and localizing performance faults in data centers, while minimizing the instrumentation intrusiveness and overhead. We show that effective monitoring can be performed by instrumenting only 21% of the components. We show that the proposed self-adaptive fault detection algorithm detects faults with low false positives and negatives. We show that the proposed signature matching algorithm performs fault localization with a reasonably small number of false negatives. The experiment results show that Varanus can perform localization with low instrumentation and greater than 80% accuracy with false positives and false negatives being less than 20%.

References

[1] E. Cecchet, M. Natu, V. Sadaphal, P. Shenoy, and H. Vin, "Performance debugging in data centers: Doing more with less," in *First International Conference on Communication Systems and Networks, Bangalore, India, January*, 2009.

[2] M. Y. Chen, E. Kiciman, E. Fratkin, O. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic internet services," in *In Proceedings of the International conference on Dependable Systems and Networks, pages 595604*, 2002.

[3] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, "Magpie: Online modeling and performance-aware systems," in *9th conference on Hot Topics in Operating Systems, Berkeley, CA, USA, HOTOS'03*, 2003.

[4] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," in *OSDI*, 2004.

[5] A. M. Glenberg and M. E. Andrzejewski, "Learning from data: An introduction to statistical reasoning," in *Lawrence Erlbaum*, 2007.

[6] DagGen, "http://www.loria.fr/ suter/dags.html,"

[7] http://www.mesquite.com/, "Csim 20 development toolkit for simulation and modeling,"

[8] 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.

[9] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed diagnosis in enterprise networks," in *SIGCOMM*, 2009.

[10] A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao, "Towards automated performance diagnosis in a large iptv network," in *SIGCOMM*, 2009.

[11] K. Appleby, G. Goldszmidt, and M. Steinder, "Yemanja-a layered event correlation system for multi-domain computing utilities," *Journal on Network and Systems Management*, vol. 10, no. 2, pp. 171–194, Jun., 2002.