Semantic-Driven Model Composition for Accurate Anomaly Diagnosis

Saeed Ghanbari and Cristiana Amza
Department of Electrical and Computer Engineering
University of Toronto
{saeed,amza}@eecg.toronto.edu

Abstract

In this paper, we introduce a semantic-driven approach to system modeling for improving the accuracy of anomaly diagnosis. Our framework composes heterogeneous families of models, including generic statistical models, and resource-specific models into a belief network, i.e., Bayesian network. Given a set of models which sense the behavior of various system components, the key idea is to incorporate expert knowledge about the system structure and dependencies within this structure, as meta-correlations across components and models. Our approach is flexible, easily extensible and does not put undue burden on the system administrator. Expert beliefs about the system hierarchy, relationships and known problems can guide learning, but do not need to be fully specified. The system dynamically evolves its beliefs about anomalies over time.

We evaluate our prototype implementation on a dynamic content site running the TPC-W industry-standard e-commerce benchmark. We sketch a system structure and train our belief network using automatic fault injection. We demonstrate that our technique provides accurate problem diagnosis in cases of single and multiple faults. We also show that our semantic-driven modeling approach effectively finds the component containing the root cause of injected anomalies, and avoids false alarms for normal changes in environment or workload.

1 Introduction

As system complexity grows, deploying automated tools for anomaly detection and diagnosis in large scale multi-tier systems has become a life-saving goal for the computer industry. Many commercial tools for coordinated monitoring and control of large scale systems exist. HP's Openview and IBM's Tivoli products collect and aggregate information from a variety of sources and present this information graphically to operators. However, the complexity of deployed systems exceeds the ability of humans to diagnose and respond to problems rapidly and correctly [1].

The traditional approach to automated problem detection is to develop *analytical* models of system structure and behavior, which may be represented quantitatively or as a set of event-condition-action rules [2]. These models may be costly to build automatically or may require extensive knowledge about the system. If specialized domain knowledge is used, these models may either be incomplete, hence inaccurate, or may become obsolete as the system changes or encounters unprecedented situations.

In contrast, recent research [3] [4] [5] [6] [7] [8] has investigated developing statistical models for automatic fault detection. These techniques derive probabilistic relationships, called functional invariants or correlations, between metrics captured at different points across the system. These approaches are generic and need little or no domain knowledge. The system raises a user-level alarm whenever it detects a significant change for one or more of these functional invariants. These approaches can be applied to a wide variety of systems, and can adapt rapidly to system changes. On the downside, they may trigger unacceptable levels of false alarms for benign changes, such as a workload mix change or an environmental setting change. Moreover, when a fault occurs, even if the fault is localized to one component, metrics collected at many other components may show abnormal functional correlations. As we will experimentally show, classifying or ranking techniques for metrics involved in correlation violations are not sufficient for pinpointing the fault location.

In this paper, we introduce a novel system modeling technique, which combines the benefits of the above approaches, while avoiding their pitfalls. The basis of our technique is a unifying framework for deploying heterogeneous types of models, each model tagged with semantic meaning corresponding to a particular system component or input. Each of these models uses the most appropriate approach, e.g., a generic analytical model, a statistical model or a fully specialized model, for accurately modeling the respective component or input. Our technique is flexible and allows for plug-and-play replacement of any per-component model. More importantly, we use a meta-model, called a Belief Network, to detect and diagnose anomalies based on the collective information gathered from all individual models. Our meta-model incorporates two key elements:

- Readily available expert knowledge about the semantic meaning and relationships between the system components, and between each component and its modeled sub-components. This information encodes the hierarchical structure of the system, including its inputs, such as environment or load, as well as their relationships.

- A Bayesian network, which makes probabilistic inferences for detecting anomalies and diagnosing the system component or input most likely to be the root cause of an observed anomaly. The Bayesian network builds on the expert knowledge of the system structure, as well as on off-line or on-line training with automated fault injection in order to form and dynamically evolve its beliefs about faulty components.

Our approach requires only high level information about the system, which typically already exists and can be easily specified. The ability to associate semantic meaning to models sensing anomalies in system components, environment or load inputs, respectively, is crucial for an accurate and meaningful anomaly diagnosis.

Expert knowledge about the *meta-level* relationships between system components, called *prior beliefs*, is desirable for enhancing the ability of the Belief Network to distinguish relevant events from noise and for filtering out false alarms. An example of a useful *meta-level* relationship is intuitively as follows: *"The workload causally affects the tier(s) it feeds"*. A similar meta-level correlation would be used to detect the influence of the environment, defined as configuration files, tuning knobs, and scheduled applications on various system components.

Prior beliefs can either be specified in a graphical way, by connecting components in a directed acyclic graph (DAG) through flow arcs depicting structural or input relationships, or in a high-level language, such as SML [9]. The availability of prior beliefs shortens or even circumvents the need for training the Bayesian Network. However, expert information does not need to be completely specified or fully accurate. Our technique enables adding new expert knowledge incrementally, and automatically prunes out or adjusts outdated beliefs about the system.

In our experimental evaluation, we use a standard dynamic Web site built using the three popular open source software packages: the Apache web server [10], the PHP web-scripting/application development language [11], and the MySQL database server [12] running the shopping and browsing mix workloads of the TPC-W e-commerce benchmark [13], an industry-standard benchmark that models an on-line bookstore.

We define a system structure consisting of the Workload, Web server, Database server tiers, several resources in each tier, such as CPU, memory, disk, network, etc. We use a specialized model for estimating the memory footprint, based on the Miss Ratio Curve (MRC) [14]. We use Gaussian Mixture Models [4] for modeling other parts of our system. We train the various models using unsupervised learning and our Bayesian network using supervised learning, through automated fault injection of a variety of fault types including disk and network hogs on system components.

Our results show that our anomaly detection method can successfully disambiguate between faults and workload or environment change cases. Our results also show that our approach can accurately pinpoint the components that are most affected by an anomaly for experiments with single or multiple faults injected into the system on-the-fly.

The rest of this paper is organized as follows. Section 2 motivates our approach through preliminary experiments using generic statistical models and introduces background on Bayesian networks. Section 3 describes the architecture of our system and introduces our Bayesian learning technique. Section 4 describes our prototype and testbed, while Section 5 presents the results of our experiments on this platform. Section 6 discusses related work and Section 7 concludes the paper.

2 Background and Motivation

In this section, we first motivate our scheme through preliminary anomaly detection experiments with generic statistical models. We then introduce necessary background on Bayesian network.

2.1 Motivating Example

In this section, we show that a generic statistical model, which tracks correlations for metrics across the whole system cannot reliably pinpoint the faulty component. We use the common three-tier architecture of a dynamic content Web site, consisting of a front-end web server, an application server implementing the business logic of the site, and a back-end database. The web site is running the shopping mix of the e-commerce TPC-W benchmark. We instrument the system to periodically sample various CPU, disk, network, and Operating System metrics for each tier. We use a model similar to Gaussian Mixture Models(GMM) [4] to approximate the probabilistic distribution of pair-wise correlation between metrics, and associate a *fitness score* [15], i.e., a confidence score, to each correlation.

We perform a fault injection experiment where we hog the disk on the machine running the database server. Our GMM model reports 122 violations among 36 metrics across the whole system. We then rank the metrics based on the number of correlation violations they appear in. Table 1 shows the top 14 ranked metrics, and the number of correlation violations they appear in.

Table 1. **Top 14 metrics involved in violations for an injected disk hog. None of the disk related metrics are among the top affected metrics.**

Rank 1 [violations:20]:DB_OS_in
Rank 2 [violations:18]:DB_NET_rx_packets
Rank 3 [violations:16]:WEB_NET_tx_packets
Rank 4 [violations:14]:DB_NET_tx_packets
Rank 5 [violations:14]:WEB_NET_rx_packets
Rank 6 [violations:10]:DB_OS_cs
Rank 7 [violations:8]:DB_OS_id
Rank 8 [violations:8]:DB_Conn_Times
Rank 9 [violations:8]:DB_IncomingQuery
Rank 10 [violations:8]:WEB_OS_free
Rank 11 [violations:7]:DB_OS_r
Rank 12 [violations:7]:DB_Latency
Rank 13 [violations:7]:DB_QueryThput
Rank 14 [violations:7]:DB_IncomingRD

Each metric is named specifying the tier, the component within the tier and the particular metric captured for that component. For example, the *DB_NET_tx_packets* metric represents the number of packets transmitted by the network from the database tier.

As we can see from the table, the top ranked metrics are not directly related to the most affected component, which is the database disk. The top ranked metric, the DB_OS_in (the number of interrupts measured at the OS component on the database tier), appears in 20 violated correlation with other metrics. We further see several highly ranked metrics related to network transmit and receive packets in both the Web and Database tiers. The disk metrics do not appear in the top correlation violations mainly because of low *fitness scores*.

2.2 Discussion of Results and Goals

The results presented show that a fault injected in one component affects system metric correlations system-wide when generic statistical models of the whole system are employed. We also show that there is no reliable method for pinpointing the faulty component when using such models.

While tagging the metrics collected with semantic meaning and ranking them based on the number of violations they are involved in may help in some cases, as was shown in previous work [4], this is not always the case. There is no simple method for associating a confidence value with a ranking result. Hence, a high level of false alarms may occur with such anomaly detection methods.

In this paper, we address this issue by providing a method for guiding learning for the purposes of improving the accuracy of fault diagnosis in terms of both i) lowering the occurrence of false alarms and ii) accurately zooming in to the faulty component through a semantic-driven modeling approach based on Bayesian Networks.

2.2.1 Bayesian Networks

A Bayesian network is a graphical model for probabilistic relationships among a set of variables [16], where dependencies among the variables are represented by a *Directed Acyclic Graph*. For a set of variables $X = \{x_1, x_2, \ldots, x_N\}$, a Bayesian Network encodes a set of conditional probability distribution of the form $p(x_i \mid pa_i)$, where pa_i is the set of parents of variable x_i. Given the DAG S and the set of conditional probabilities, the joint distribution for X is given by $p(X) = \Pi_{i=1}^{N} p(x_i \mid pa_i)$.

Once the Bayesian network evolves its beliefs based on prior knowledge, data, or both, we can determine various probabilities of interest from this model. The computation of a probability of interest given a model is known as probabilistic inference.

In conjunction with statistical techniques, Bayesian networks have several advantages for inference under uncertainty. Bayesian networks encode dependencies among variables, hence are robust to situations where some data entries are missing or the data sampling is noisy. The other advantage of a Bayesian network stems from the fact that it models both causal and probabilistic semantics, which makes it an ideal representation for combining prior knowledge, which often comes in causal form, and experimental data [16].

3 Semantic-Driven System Modeling Using a Belief Network

In this section, we describe our approach to semantic-driven system modeling in a three tier dynamic content web site. Our assumption is that a single component fault may occur at any given time, but with potential manifestations in other components. Our objective is to accurately pinpoint the component or system input most likely to contain the root cause of an anomaly. Towards this goal, we use a hierarchical modeling approach called a *belief network* combining i) heterogeneous models acting as system sensors, one per identifiable system component or input and ii) a meta-model incorporating expert knowledge into Bayesian learning.

In the following, we first introduce the overall architecture of our system and an overview of our approach. Then, we describe how the human interacts with the system by i) providing expert knowledge about system structure and the correlations between components, and ii) querying our belief network for anomaly diagnosis. Finally, we describe our meta-model based on Bayesian learning.

3.1 Overview

Our fault diagnosis belief network is a hierarchical network, structured as a directed acyclic graph (DAG), reflecting the underlying system architecture and the correlations between components. Figure 1 shows part of our layered structure belief network for dynamic content servers, which is built from the following elements: inputs, tiers, components, sensor models, and the correlations across them as follows.

The environment parameters per tier, and the workload represent **inputs** to the rest of the system. A node in the network represents each **tier** in a multi-tier system, i.e., T_{WL} - the client/workload tier, T_{Web} - the web tier and T_{DB} - the database tier. The **component** layer in the network models the hardware and/or software components belonging to each tier, e.g., the disk (C_{DiskDb}), network (C_{NetDb}), etc. At the lowest level, we have a set of **sensor models** monitoring different system sub-components, each tagged with a clear semantic meaning, e.g., M_{SysDb} and M_{DbIO} are models sensing different aspects of the disk component of the database tier. Each sensor model is a fully fledged anomaly detection model, which expresses its confidence about detecting an anomaly in its monitored sub-component. The sensor models provide evidence of per-component anomalies for all upper layers of the belief network. A **correlation** between two nodes in the network, shown as a directed arc, represents a dependency between the probability of anomaly of the two nodes. A correlation can have one of the following semantic meanings: i) an *input* correlation encodes the fact that an input change is likely to cause changes within the tier(s) it feeds, ii) a *structural* correlation encodes the association between a tier and its components or a component and its sub-components and iii) a *causal* correlation encodes a cause-and-effect relationship between components or between a known problem and the components affected by that problem. For example, an anomaly in the cache component at the database will likely cause the disk component on that tier to be affected.

3.2 Operation

We use Bayesian learning to infer the probability of anomaly at each node in our belief network. Bayesian networks provide a simple way to encode prior knowledge. Prior knowledge in our case is semantic structure, such as the multi-tier nature of the system, an expert's belief about the relationships of interacting components, or known problems, such as, recurrent faults and the components that are affected by such faults.

The network, which encodes semantics and initial expert beliefs about dependencies between components is trained with data collected from the working system in experiments using fault injection. The expert knowledge about inter-node relationships makes the network converge faster during training and filters out false alarms and noise in the system. Conversely, in the process of training, initial beliefs that do not match real data are eliminated, while the belief network gains stronger confidence in valid initial beliefs. The belief network learns the statistical dependencies among components and models. It then calculates a probability of fault occurrence at each node, at any given time, based on evidence of anomalies from sensor models.

We employ a metric collection system to provide data for the various types of sensor models, including system metrics, such as network traffic statistics, OS statistics, including CPU and memory utilization, and software component statistics, such as, database statistics logs.

In the following, we describe how the system administrator interacts with our belief network by specifying expert knowledge and by querying the network for anomalies. We then describe our models and the Bayesian learning technique in this context.

3.3 Expert Knowledge Specification

The system administrator specifies dependencies between system components as arcs, each weighted by a *dependency value*. Each arc represents a correlation between the probabilities of anomaly of the two nodes involved. The expert can be conservative by providing a superset of arcs. The system will prune out irrelevant correlations during training. Likewise, the initial *dependency value* on each arc serves only as a guidance and is adjusted based on data during training.

Each dependency value indicates either a *direct correlation*, typically associated with structural correlations, or an *inverse correlation*, usually corresponding to input and causal correlations.

As an example of a direct correlation, a high probability of fault occuring within a tier/component will be associated with a high probability of fault within one or more of its components/sub-components, respectively. As an example of an inverse (input) correlation, if the probability of an input change is high, the probability that the changes perceived within the Web tier indicate a fault originating within that tier is low. Likewise, if an anomaly occurs in the database cache miss rate, any anomalies perceived in the DB disk are likely not due to a fault in the DB disk itself; hence the probability of fault in the DB disk should be low.

3.4 Querying the Network for Anomaly Diagnosis

Each node in the belief network computes a probability of fault/anomaly, which we call *confidence*, originating

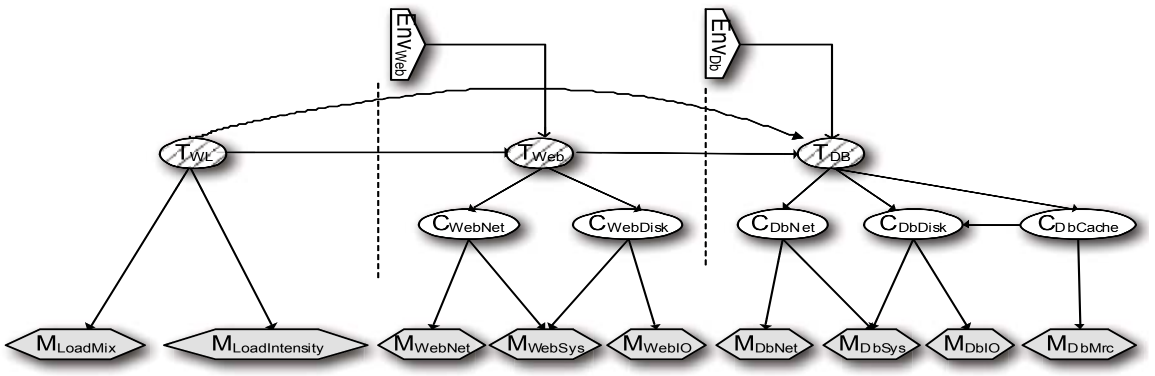

Figure 1. **Example Belief Network**

within the part of the system it represents; in our example, T_{Web} and C_{DiskDb} compute probabilities of anomaly originating from within the Web tier, and from within the disk component of the database, respectively.

Querying the belief network for the purposes of anomaly diagnosis proceeds top-down through the hierarchy embedded through dependencies in the belief network. If all input and tier nodes of the belief network show a low probability of anomaly, then no anomaly is present, regardless of any alarms triggered by the low-level sensor models.

If the workload tier or an environment node shows a high probability of anomaly, then no other tier would likely show a high probability of anomaly at the same time, due to the respective inverse (input) correlations. In the absence of any workload or environment anomalies, diagnosis continues by querying the lower levels as follows. If a tier shows a high probability of anomaly, then an anomaly is detected within that tier and per-component anomalies of that tier are queried. A high probability of anomaly within a single tier, coupled with a high probability of fault within a specific component of that tier pinpoints that component as containing the root cause of the anomaly.

3.5 Belief Network Meta-Model with Bayesian Learning

In its inferences about the per-node probability of anomaly, our belief network uses Bayesian learning of per-node fault probabilities guided by an encoding of correlations between tiers, components and models.

The distribution over the confidence values of each model $M_i \in R$ is treated as a Gaussian distribution $N(M_i|\mu_{pa_i}, \sigma_{pa_i})$, where the mean and variance are conditioned by the state of the parents of M_i in the belief network.

The belief network builds on both the cumulative information of the sensor models and expert knowledge about the relationships between components. In this section, we first describe how we encode the various nodes in the Bayesian network and the relationships between them. Then we describe our Bayesian supervised learning scheme for evolving probabilistic beliefs about anomalies in the system based on training with fault injection.

3.5.1 Encoding Tiers, Components and Inputs

The state of a component, or tier is represented by a two-state variable $C_i \in \{anomalous, healthy\}$ with Bernoulli distribution of the form $p(C_i = anomalous|pa_i) = \theta_{pa_i}$, and $p(C_i = healthy|pa_i) = 1 - \theta_{pa_i}$, where pa_i is the state of C_i's parents. The belief network can also encode known problems which are represented by Bernoulli variables $K \in \{yes, no\}$. For example, if an expert knows about a probable anomaly in processing PHP code, our belief network can encode her beliefs about which components and which models might be affected e.g., a model sensing the distribution of queries at the database tier.

Finally, our technique incorporates input variables, as workload and environment variables to make the belief network aware of external changes to the system and their resulting dependencies. Input variables represent external changes such as load intensity, load-mix changes or configuration file modifications, interference due to scheduling of a third party application on a tier, etc. Changes in workload and environment are represented by Bernoulli variables WL and an ENV_{tier} variable per tier, respectively, which take the values $\{changed, unchanged\}$.

3.5.2 Encoding Dependences between Variables

Expert knowledge is encoded as dependencies between variables and their dependency values. The structure is expressed as a directed acyclic graph (DAG). To incorporate initial expert belief about the degree of dependency of two Bernoulli variables, a prior belief with beta distribution $\beta(\alpha_{pa_i}^{anomalous}, \alpha_{pa_i}^{healthy})$ is used. The higher the $\alpha_{pa_i}^{anomalous}$

relative to $\alpha_{pa_i}^{healthy}$, the stronger the a priori belief that a component is more likely to be anomalous for a state of anomaly in its parents in the DAG, pa_i. Conversely, if $\alpha_{pa_i}^{anomalous}$ is substantially lower relative to $\alpha_{pa_i}^{healthy}$, then a component is unlikely to be anomalous for a given state of anomaly in its parents. We use the former type of prior belief to encode parent-child dependences between tiers and their components and between components and models. We use the latter type of prior belief to suppress false alarms due to anomalies in input. In the latter case, our belief network will automatically lower the confidence of a fault detection of a tier node if an input that feeds it, either another tier, or an external input i.e., workload or environment, shows an anomaly.

3.5.3 Training Using Fault Injection

Once the structure and initial degrees of dependency among components are determined by the expert, the resulting belief model is ready for training. Training has two phases: training the models, and training the belief network. To conduct the supervised training of our belief network, we perform automatic fault injection into components to observe its impact on models.

Upon fault injection, the parameters of the Gaussian distribution of a variable M_i evolve according to the following equations:

$$\mu_{pa_i} = average(M_i \mid pa_i), \sigma_{pa_i}^2 = variance(M_i \mid pa_i) \quad (1)$$

where $M_i \mid pa_i$ is the observed value of M_i when the state of its parents is pa_i.

The parameters of a Bernoulli variable C_i are computed as follows:

$$\theta_{pa_i} = \frac{N_{pa_i}^{C_i=anomalous} + \alpha_{pa_i}^{anomalous}}{N_{pa_i} + \alpha_{pa_i}} \quad (2)$$

where $N_{pa_i} = N_{pa_i}^{C_i=anomalous} + N_{pa_i}^{C_i=healthy}$ and $\alpha_{pa_i} = \alpha_{pa_i}^{anomalous} + \alpha_{pa_i}^{healthy}$. $N_{pa_i}^{C_i=anomalous}$ is the number of times variable C_i is anomalous given the state of its parents pa_i, and $alpha_{pa_i}$ is the prior belief which encodes the initial expert belief about the state of anomaly of C_i given the state of its parents. The parameters of all other Bernoulli variables, i.e. T_{tier}, ENV_{tier}, and K are computed in a similar manner, according to Equation 2.

3.5.4 Inferences within the Belief Network

Once the belief network is trained, it can infer the state of any component, tier, or occurrence of a known problem given the state of all the individual models. The inference about the degree of anomaly in a component, C_i, is made by computing the posterior probability $P(C_i = anomalous \mid M)$, where M is the observation of the states of *all* the models. We use the Variable Elimination algorithm [17] to compute the posterior probability for variables of interest.

3.6 Discussion of Trade-Offs

The granularity of anomaly diagnosis directly depends on the level of detail of the system hierarchy description and the availability of models to sense each component and input of this hierarchy. For example, the belief network cannot distinguish an anomaly in the database server cache manager versus an anomaly within the application running on the database server, unless specific sensors for each are predefined. Hence, there is a trade-off between the burden on the system administrator for specifying expert knowledge and providing the relevant sensor models on one hand, and the usefulness of the results when querying the belief network on the other hand. However, our approach can be incrementally refined by gradually adding new models and relationships to the hierarchy. Accurate initial estimates of the degree of dependencies between nodes in the network produces fast convergence during training. However, as we will show, the network is resilient to human error and corrects any initial errors, such as extra links or inaccurate dependency values on links during training.

4 Prototype and Testbed

We implement the Bayesian inference engine, and all other models in Java. We use two types of models as our system sensors: a specialized resource model based on the Miss Ratio Curve (MRC) for sensing memory footprint anomalies, and Gaussian Mixture Models (GMM) for all other system components. We describe the MRC and GMM models next.

MRC Model: The miss-ratio curve (MRC) of an application shows the page miss-ratios at various amounts of physical memory. This approach was first used in cache simulation and was recently proposed for dynamic memory management [18]. The MRC reveals information about an application's memory demand, and can be used to predict the page miss ratio for an application, given a particular amount of memory. The MRC can be computed dynamically through Mattson's Stack Algorithm [14] . The algorithm is based on the inclusion property, which states that a memory of k + 1 pages includes the contents of a memory of k pages. The popular Least Recently Used (LRU) algorithm exhibits the inclusion property, thus allowing us to estimate the miss ratio for a given amount of memory *m*.

We employ the MRC to model the memory footprint of the database server. We sample the MRC at various points in time and we compute the vector of mean($\mu_{mrc(i)}$) and variance ($\sigma_{mrc(i)}$) of the sampled curves for $0 \leq i \leq e$, where e

is the effective size of memory. We use the following equation to measure the proximity of the observed MRC to the expected/normal curve:

$$dist = \frac{1}{e}\sum_{i=0}^{e}\frac{(MRC(i)-\mu_{mrc}(i))^2}{\sigma^2_{mrc(i)}} \tag{3}$$

Gaussian Mixture Models: Gaussian Mixture Models (GMM), a clustering algorithm, is used to capture the statistical correlation between pairs of metrics [3, 4].

In contrast to previous work on GMM [4] which determines statistical correlations between all pairs of metrics, we break down metrics into subsets, each subset corresponding to a specified system part. Each model represents a semantically meaningful system part; it derives pairwise correlations among the subset of metrics belonging to its respective system part, as well as between this subset of metrics and all other metrics. This enables us to model behavior of components of interest. For example, to model the disk I/O, we derive the correlations between metrics pertaining to disk as well as correlations between disk metrics and all other metrics. Collectively these correlations form the basis for the disk I/O model, which raises alarms with a certain confidence if any of these correlations are violated. In our experiments, we use GMM to model Disk I/O and Network.

Training the Models: Training of each model can be either supervised or unsupervised, depending on the type of the model. In this paper, we train each model individually, in an unsupervised manner. The size of the data needed for each model is also different. For example, the MRC model typically uses a trace of 1 million page accesses, which might take a couple of minutes to collect, while GMM-based models need a set of samples collected over a 30 minute period, on average, in order to achieve a stable model.

Testbed and Statistics Collection: We use the industry-standard TPC-W e-commerce benchmark [13] running the browsing and shopping workload mixes. TPC-W is implemented using three popular open source software packages: the Apache web server [10], the PHP web-scripting/application development language [11] and the MySQL database server [12]. We use the Apache 1.3.31 Web server in conjunction with the PHP module. We use MySQL 4.0 with InnoDB tables as the database back end. We use a client emulator which drives our experiments. Our experimental testbed is a cluster of Xeon Intel computer with 2GB of RAM and four 3.00GHz CPUs. All the machines use the Ubuntu Linux operating system. All nodes are connected through 1Gbps Ethernet LAN.

We use lightweight tools for measuring system-level and application-level metrics. Monitored metrics include OS, network, database and workload statistics. The data is obtained from `vmstat`, `netstat`, instrumented `Mysql`, and the workload emulator. We use a scheduler interposed between Apache-PHP and Mysql to collect metrics of interest about queries and database statistics, such as throughput and active connections. Logs obtained from the client emulator are used to model workload characteristics. We employ a central data collection layer which collects metrics from various points in the system online, and after preprocessing provides it to the models. To facilitate data collections, daemons residing on each tier, and inside applications of interest collect data and send it to the centralized data collection layer.

5 Experiments

In this section, we validate our anomaly diagnosis approach through several experiments. We first conduct experiments with a single system-level fault or workload mix change, then with multiple (two) simultaneous system-level faults showing the diagnosis capabilities of our system. Finally, we induce an application-level change unknown to the system, and show how our system detects the most affected component.

The structure of the belief network used in these experiments is shown in Figure 1. We consider five components in our experimental belief network: $C_{WebDisk}$, C_{WebNet}, C_{DbDisk}, C_{DbNet} and $C_{DbCache}$ i.e., the disk, and network on the web tier, and the disk, network and database cache, on the database tier, respectively. We use 8 models. Except for M_{DbMrc}, modeling the database cache, all models are based on GMM.

Single Fault Diagnosis. We train and evaluate the belief network with injecting two types of faults: network hogs, and disk hogs. In addition, we induce load mix changes. Network hogs are induced by flooding the network with useless packets to a victim machine. A disk hog is implemented by spawning a thread running the Unix command `dd` on a victim machine, which reads a large file and copies it to the `/tmp` directory. We induce disk and network hogs on the Web and database machines, in separate experiments. We also induce a load mix change, from browsing to shopping, while only the browsing mix is used during training. Figure 2 shows the reaction of all 8 low-level sensor models to each type of fault. Table 2 shows the corresponding results of fault diagnosis at the upper levels in the belief network for each type of fault, respectively. Each row in the table corresponds to a fault type. Each element in the row indicates the confidence/probability of fault in either a tier or component corresponding to each column of the table. This data shows that our scheme can diagnose the anomalous component with high degree of confidence in each case of anomaly. The table also shows that, when there is no anomaly in the system, all tiers indicate a low probability of fault.

The results of the experiment for a load mix change show the ability of our scheme to avoid false alarms in cases of

Table 2. Single fault diagnosis

Fault	T_{WL}	T_{Web}	C_{WebNet}	$C_{WebDisk}$	T_{Db}	C_{DbDisk}	C_{DbNet}
Db Disk	0	0.03	0	0	0.91	1	0
Db Net	0	0.03	0	0	0.91	0	1
Web Net	0	0.91	0.99	0	0.04	0	0
Web Disk	0	0.91	0	1	0.04	0	0
Load Mix	0.99	0.04	0.02	0	0.40	0.99	0
No Fault	0	0.25	0	0	0.25	0	0

an unencountered, but benign change in workload. As we can see from Figure 2(LoadMix), out of the 8 models, the $M_{LoadMix}$ and M_{DbSys} models show the highest probability of anomaly. Correspondingly, in table 2(Load Mix), C_{DbDisk} (database disk) shows a high probability of anomaly. However, since the dependencies between tiers have been encoded in the network, the database tier shows a relatively low (40%) confidence in detecting an anomaly originating within that tier. Hence, our scheme diagnoses the change in the workload, as detected by T_{WL}, as the reason for the change in behavior seen at the database disk.

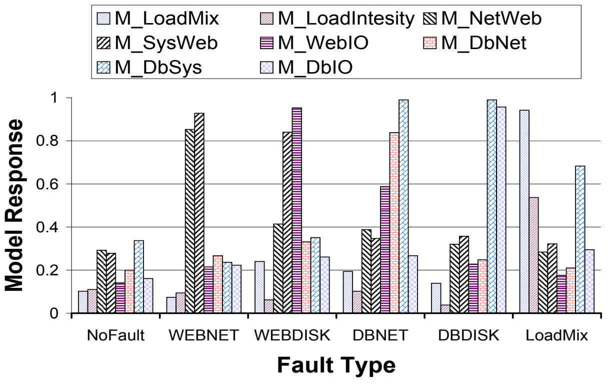

Figure 2. Model responses to various faults

Multiple Simultaneous Faults. We inject two of the disk and network hogs simultaneously, on-line, while the network is trained using single fault injection as before. The diagnosis is difficult because the faults manifest in different sensor models. Table 3 shows the fault diagnosis of our network for all cases. As expected, the confidence at the tier-level is very high when both faults are injected in the same tier. In the case of fault injection into different tiers, the dependency between tiers results in a 56% confidence at the tier level for both tiers i.e., the belief network cannot determine the tier where the fault originates from, as expected, but the belief network has high confidence at the component level.

Unknown Fault. In this experiment, we show the ability of our model to detect a type of fault not seen during training, i.e., an induced anomaly in the memory footprint of the database cache (modeled using the MRC model). We substantially change the pattern of the frequent *Bestseller* query in the TPC-W benchmark. The new query pattern increases the memory footprint of the frequent query, which however still fits in the database cache. In this experiment, we disable models directly sensitive to the root cause of this fault. This includes disabling all models sensing environment changes and query access pattern changes within the workload, which would otherwise pinpoint the top level cause of anomaly to an input/configuration change as in the load mix change experiment.

Table 4. Problem diagnosed in the DB tier within the DB cache

Component	*Confidence*
T_{Db}	**0.93**
$C_{DbCache}$	**1.00**
C_{DbNet}	0.61
C_{DbDisk}	0.00
T_{Web}	0.64
C_{WebNet}	1.00
$C_{WebDisk}$	0.00

Due to the frequency of the query, the anomaly manifests itself in several components of both tiers. For example, the database cache is affected, but the network on the Web tier is affected as well, due to an increase in the response size of this query. Almost all model responses record a normalized deviation between 30% and 150% from the average NoFault situation, which makes diagnosis of the faulty component non-trivial.

Overall, as we can see from Table 4, our belief network makes a diagnosis of fault within the database tier, rather than the web tier in this case. The intuition behind the higher fault probability at the database tier is that two of the components at the database tier show an anomaly. Moreover, the memory footprint of the *Bestseller* query is very stable during training. In contrast, the recorded statistical variation of all other models is substantially higher. Since inferences are made based on training data, as well as evidence of fault from *all* sensor models, our belief network identifies the fault as occurring within the database tier, specifically within the database cache ($C_{DbCache} = 1.0$). This pinpoints the component most affected by the root cause of the

Table 3. **Diagnosis in case of simultaneous faults**

Fault	T_{Web}	C_{WebNet}	$C_{WebDisk}$	T_{Db}	C_{DbDisk}	C_{DbNet}
Web Disk, Db Disk	0.56	0.99	0	0.56	1	0
Web Net, Web Disk	0.99	0.99	0.99	0.01	0	0
Web Disk, Db Net	0.56	0	1	0.56	0	1
NoFault	0.25	0	0	0.25	0	0

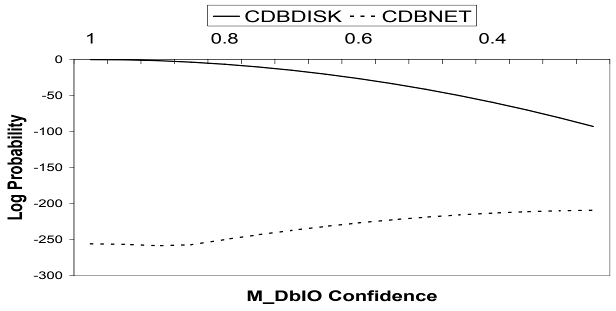

Figure 3. **An invalid initial belief is pruned after training**

anomaly, hence providing valuable information in the absence of the root cause itself, which the system cannot detect in this case.

Pruning Irrelevant Dependencies. In this experiment, we show that our network is resilient to mistakes of the human expert or any irrelevant initial links. We define an (invalid) dependency between the C_{DbNet} component and the database I/O model (M_{DbIO}) and re-train the belief network. After re-training, Figure 3 shows that the network has automatically modified the dependency value between C_{DbNet} and M_{DbIO}. To show this, we plot the inferred log probability of anomaly in the disk and network, C_{DbDisk} and C_{DbNet}, while decreasing the probability of fault (confidence) as sensed by the database I/O model from 1 to 0. We can see that the inferred probability of fault in the DB network (C_{DbNet}) is very small (almost zero, due to the log scale), regardless of the probability of fault detection read from M_{DbIO}. This is very different from a component with a true dependency to M_{DbIO}. We show the variation of the probability in C_{DbDisk} as measured during the same experiment, which decreases significantly with the decrease in probability in M_{DbIO}. This shows that after training, a wrong link is pruned by automatically adjusting the degree of dependency to a value which no longer influences the inference at C_{DbNet}. This reduces the burden of the expert by allowing for placing extra (default) links, or random dependency values on links in the initial network.

6 Related Work

Jiang *et al.* [3] and Chen *et al.* [5] introduce an automatic fault diagnosis scheme based on pairwise statistical correlation of monitoring data. They define the concept of *invariant* as a correlation between a pair of metrics which holds under various load intensities. Invariants are used as the basis of anomaly detection. They introduce a ranking scheme to diagnose the most affected component in case of anomaly. Unlike our scheme, invariants lack semantic attribution, which as we have shown, may make diagnosis difficult. Invariants are also vulnerable to changes in workload mix, which would result in a high rate of false alarms.

Pinpoint [7, 19, 20] and Magpie [21] are statistical tools for fault detection in component-based Internet services. Both tools model component interactions. Unlike our scheme, they do not adapt their model dynamically, and do not leverage semantics or expert knowledge, hence may experience a high rate of false alarm rates for unexpected user behavior or workload mixes.

Cohen *et al.* [22] introduce an automated method to correlate system-level metrics to high-level system states. The key idea is to deploy a graphical statistical model, Tree Augmented Bayesian Networks (TAN), to characterize high level system state in terms of their dependency to low level metrics. The TAN graphical model provides insights into the underlying cause of faults. Similarly, Zhand *et al.* [23], and Kumar *et al.* [24] use an ensemble of TAN models to capture various system states. These ensembles are used to derive signatures of a service for the purposes of anomaly detection [25]. TAN based anomaly detection shares similarities with our scheme. However, our scheme has the advantage of encoding expert knowledge directly in the model. Also, while this approach uses an ensemble of models of the same type, i.e., TAN, our scheme incorporates heterogeneous types of models.

PeerPressure [26] is an automatic misconfiguration troubleshooting tool using Bayesian inference and statistical reasoning based on peer configuration settings. Yuan *et al.* [27] identify known problems by mapping traces of function calls to known high level problems. Both schemes view the system as a black box and do not leverage expert domain knowledge or semantics for fault diagnosis.

7 Conclusions

In this paper, we introduce a novel semantic-driven approach to anomaly detection and diagnosis. Our approach bridges and integrates heterogeneous per-component models through a Bayesian network encoding the semantic meaning of components, their interactions and the association between components and models.

We train the underlying models through unsupervised learning. The Bayesian network trains itself by observing the reaction of each model to injected faults. Both per-component sensor models and the Bayesian meta-model thus evolve dynamically as the system or application changes. The close correspondence between the structure of the meta-model and the structure of the system facilitates diagnosis of the component that most likely contains the root cause of a perceived anomaly. This allows us to differentiate component faults from cases of external changes of workload or environment.

We perform experiments using the industry-standard TPC-W benchmark and a wide range of fault injection scenarios, including disk, network and application anomalies, as well as single and multiple faults. We show that our approach provides accurate diagnosis, while requiring minimal human expert intervention. Providing readily available information on system structure and dependencies results in rapid convergence during training. Conversely, the network is resilient to human error and prunes out or adjusts incorrect dependency information.

Acknowledgments

We thank the anonymous reviewers for their comments, and acknowledge the generous support of IBM Centre for Advanced Studies (CAS), IBM Research, the Natural Sciences and Engineering Research Council of Canada (NSERC), Ontario Centers of Excellence (OCE), and Canadian Foundation for Innovation (CFI).

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier Internet services and its applications." in *SIGMETRICS'05*, 2005, pp. 291–302.

[3] G. Jiang, H. Chen, and K. Yoshihira, "Discovering likely invariants of distributed transaction systems for autonomic system management," *Cluster Computing*, vol. 9, no. 4, pp. 385–399, 2006.

[4] Z. Guo, G. Jiang, H. Chen, and K. Yoshihira, "Tracking probabilistic correlation of monitoring data for fault detection in complex systems," in *DSN'06*, 2006, pp. 259–268.

[5] H. Chen and K. Yoshihira, "Modeling and tracking of transaction flow dynamics for fault detection in complex systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 4, pp. 312–326, 2006.

[6] A. Fox, E. Kiciman, and D. A. Patterson, "Combining statistical monitoring and predictable recovery for self-management." in *WOSS'04*, 2004, pp. 49–53.

[7] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. A. Brewer, "Pinpoint: Problem determination in large, dynamic internet services." in *DSN'02*, 2002, pp. 595–604.

[8] P. T. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling." in *OSDI'04*, 2004, pp. 259–272.

[9] IBM Corporation, "Service Modeling Language (SML) Specification," http://www-03.ibm.com/autonomic/industry-sml.html/.

[10] "The Apache Software Foundation," http://www.apache.org/.

[11] "PHP Hypertext Preprocessor," http://www.php.net.

[12] "MySQL," http://www.mysql.com.

[13] "Transaction Processing Council," http://www.tpc.org/.

[14] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal*, vol. 9, no. 2, pp. 78–117, 1970.

[15] G. Jiang, H. Chen, and K. Yoshihira, "Efficient and scalable algorithms for inferring likely invariants in distributed systems," *IEEE Trans. on Knowl. and Data Eng.*, vol. 19, no. 11, pp. 1508–1523, 2007.

[16] D. Heckerman, "A tutorial on learning with bayesian networks," 1995. [Online]. Available: citeseer.ist.psu.edu/heckerman96tutorial.html

[17] R. Dechter, "Bucket elimination: A unifying framework for probabilistic inference," in *UAI*, 1996, pp. 211–219.

[18] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic tracking of page miss ratio curve for memory management," in *ASPLOS-XI*, 2004, pp. 177–188.

[19] E. Kiciman and L. Subramanian, "A root cause localization model for large scale systems." in *Hot Topics On Dependability (HotDep)*, 2005, pp. 375–388.

[20] M. Y. Chen, A. Accardi, E. Kiciman, D. A. Patterson, A. Fox, and E. A. Brewer, "Path-based failure and evolution management." in *NSDI'04*, 2004, pp. 309–322.

[21] P. T. Barham, R. Isaacs, R. Mortier, and D. Narayanan, "Magpie: Online modelling and performance-aware systems." in *HotOS*, M. B. Jones, Ed., 2003, pp. 85–90.

[22] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control." in *OSDI'04*, 2004, pp. 231–244.

[23] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox, "Ensembles of models for automated diagnosis of system performance problems." in *DSN'05*, 2005, pp. 644–653.

[24] "Enabling policy-driven self-management for enterprise-scale systems," in *HotAC II*, 2007, pp. 25–34.

[25] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history." in *SOSP'05*, 2005, pp. 105–118.

[26] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang, "Automatic misconfiguration troubleshooting with PeerPressure." in *OSDI*, 2004, pp. 245–258.

[27] C. Y. N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma, "Automated known problem diagnosis with event traces." in *EuroSys'06*, 2006, pp. 375–388.