Failure Diagnosis of Complex Systems

Soila P. Kavulya[1], Kaustubh Joshi[2], Felicita Di Giandomenico[3], Priya Narasimhan[1]

[1] Carnegie Mellon University, PA, USA
spertet@ece.cmu.edu,priya@cs.cmu.edu
[2] AT&T Labs Research, NJ, USA
kaustubh@research.att.com
[3] ISTI-CNR, Pisa, Italy
felicita.digiandomenico@isti.cnr.it

Abstract Failure diagnosis is the process of identifying the causes of impairment in a system's function based on observable symptoms, i.e., determining which fault led to an observed failure. Since multiple faults can often lead to very similar symptoms, failure diagnosis is often the first line of defense when things go wrong - a prerequisite before any corrective actions can be undertaken. The results of diagnosis also provide data about a system's operational fault profile for use in offline resilience evaluation. While diagnosis has historically been a largely manual process requiring significant human input, techniques to automate as much of the process as possible have significantly grown in importance in many industries including telecommunications, internet services, automotive systems, and aerospace. This chapter presents a survey of automated failure diagnosis techniques including both model-based and model-free approaches. Industrial applications of these techniques in the above domains are presented, and finally, future trends and open challenges in the field are discussed.

1 Introduction

The issue of diagnosing hardware and software failures to find the underlying causes has existed for as long as computers have been around. Using the fault, error, and failure nomenclature of [53], failure diagnosis is the process of identifying the fault that has led to an observed failure of a system or its constituent components. In any sufficiently large computing system, many types of faults are often not directly visible for a number of reasons - either due to the characteristics of the fault itself, due to fault-tolerance mechanisms built into the system that hide the expression of the fault, or as is most often the case, the lack of detailed monitoring functionalities that can detect and report on the occurrence of the fault directly. In some cases, monitoring systems may provide only an indication that a fault has occurred, but may not provide sufficient information to precisely locate it.

Failure diagnosis is a technically challenging endeavor because the relationship between faults, failures, and their observable symptoms is a complex one;

single faults often produce multiple symptoms in different parts of a system, *e.g.*, a misconfiguration fault in a critical network component such as a Dynamic Host Configuration Protocol (DHCP) server can cause all client computers on the network to fail; conversely, similar symptoms may be caused by many different types of faults, *e.g.*, the failure of a networked computer to receive an IP address can have several causes including, but not limited to, packet loss in the physical network, a client misconfiguration, or a problem with the DHCP server. As operational systems become more mature, the failures they encounter often transition from easy to detect "hard failures" that cause a significant impairment to the system's primary function, to "soft failures" such as those due to performance bottlenecks or transient faults that are much harder to detect. Therefore, the process of diagnosis often also includes the identification of anomalous conditions that are symptoms of the occurrence of faults.

In addition to its essential role as the precursor to any remediation actions for maintaining a system's health at runtime, failure diagnosis also serves several important roles in resilience assessment of complex systems. Since it is only the symptoms of a fault that are usually observed at runtime, diagnosis is essential for the accurate cataloguing of fault occurrences in the field. Conversely, any data that reports on occurrences of actual system faults is by definition the product of a diagnostic process, whether it is a simple one (in case of a one-to-one mapping between faults and symptoms), a complex manual process, or an automatic one. Understanding this process is important for understanding the biases and limitations of the field data. Diagnosis is also important for discovering new fault types that can then be used to drive fault injection campaigns as discussed in Chapter X. In fact, diagnosis is the converse process of fault injection. In fault injection, one injects faults into a system according to a predefined fault model in order to analyze the resulting symptoms, or if the system tolerates the fault, the absence of any symptoms. In diagnosis, one infers the faults from the observed symptoms. Finally, diagnosis is also important in the emerging field of online resilience assessment as described in Chapter X. In this area, diagnosis can be used, under the label of fault localization, to infer the true health of complex distributed systems, including what components have actually failed, by eliminating those failure symptoms that are a result of error propagation to an otherwise operational part of the system.

Due to the complexity of computing systems and difficulty of formalizing the scope of the diagnosis task itself, diagnosis has historically been a largely manual process requiring significant human input. However, techniques to automate as much of the process as possible have significantly grown in importance. In domains such as communication networks and Internet services, the sheer scale of modern systems and the high volumes of impairments they face drive such trends, while in domains such as embedded systems and spacecraft, it is increasing complexity together with the need for autonomic operation (*i.e.*, self-healing) when human expertise is not available, that are the drivers. Due to the diversity of the domains, a variety of failure diagnosis techniques drawing from diverse areas of computing and mathematics such as artificial intelligence,

machine learning, statistics, stochastic modeling, Bayesian inference, rule-based inference, information theory, and graph theory have been studied in the literature. Finally, when automated techniques fail, approaches that assist humans perform diagnosis more efficiently via the use of visualization aides have also been widely deployed. While a comprehensive survey of this broad topic can provide sufficient material for a book of its own, in this chapter, we provide a summary of the most important techniques, and provide references to more in-depth surveys where available. This chapter is organized as follows: Section 2 discusses types of problem diagnosis techniques using illustrative examples; Section 3 highlights practical uses of these diagnosis techniques in industrial applications; Section 4 presents future trends and open challenges in diagnosisx; and Section 5 concludes.

2 Techniques

Automated problem diagnosis techniques localize the most likely sources of a problem to a set of metrics (*e.g.*, anomalous CPU usage), a set of nodes (*e.g.*, anomalous web server), or a type of problem (*e.g.*, using known problem signatures to identify misconfiguration). Operators use the output of automated problem diagnosis to guide root-cause analysis by analyzing source-code, or hardware and software settings at the identified culprits. For example, an examination of the source-code at the web server might show that the anomalous CPU activity at the web server was due to an infinite loop in a scheduling function. Automated diagnosis techniques are not perfect and they can either fail to detect a problem, *i.e*, false negative, or indict the wrong component, *i.e.*, false positive. These techniques rely on tuning to minimize the number of false negatives and false positives generated. Visualization tools complement automated problem diagnosis tools by allowing operators to visualize anomalies and explore different hypotheses on the root-cause of problems. Table 1 provides a summary of the techniques described in this chapter. For each technique, we first use an illustrative example to highlight its application, before delving into the different approaches proposed in the research literature. We conclude each discussion with a critique of the technique that highlights its strengths and limitations.

2.1 Rule-based

Rule-based techniques rely on expert knowledge expressed as a set of predefined directives, *i.e*. rules, to diagnose problems. The rules are typically formatted as a set of *if-then* statements where the *if-part* of the rule is called the *premise*, and the *then-part* of the rule is the conclusion. An example of a rule used for diagnosis is "*if* CPU utilization exceeds 90% *then* node is overloaded". Rule-based techniques for diagnosis typically rely on forward-chaining inference mechanisms [86] to synthesize results when multiple rules fire. Forward inference processes events, such as high CPU and memory utilization, and uses the triggered rules to draw conclusions on the root-cause of the problem.

Table 1. Summary of Diagnosis Techniques

Technique	Limitations
Rule-based techniques rely on expert knowledge expressed as a set of predefined rules to diagnose problems *(Section 2.1)*.	Rules are human-interpretable and extensible. However, they cannot diagnose unforeseen problems, and large knowledge bases are difficult to maintain.
Model-based techniques define a mathematical representation of a system, testing the observed state against the model to see if it conforms *(Section 2.2)*.	Model-based techniques are well suited for diagnosing application-level problems. However, building models requires a deep understanding of the system.
Statistical techniques summarize and interpret empirical data using techiques such as correlation, histogram comparison and probability theory, for diagnosis *(Section 2.3)*.	Statistical techniques require little expert knowledge or detailed models on system internals. However, they have difficulties distinguishing legitimate changes in behavior (*e.g.* workload changes from illegitimate changes (*e.g.* performance problems).
Machine-learning techniques identify patterns in behavior using clustering, or use training data to determine if the system is unhealthy and the likely cause *(Section 2.4)*.	Machine-learning techniques automatically learn profiles of system behavior, but can suffer from the curse of dimensionality that reduces accuracy when the number of features is large.
Count-and-threshold techniques allow discrimination between transient and intermittent faults *Section 2.6)*.	Diagnosis accuracy strongly depends on proper parameter calibration. However, solutions for parameter tuning based on rigorous mathematical formulations and analytical models are available.
Visualization techniques allow operators to visualize trends in data and spot anomalous behavior *(Section 2.5)*.	Visualization tools allow operators to explore different hypotheses on the root-cause of problems. However, they do not automatically identify they source of problems.

Illustrative example Chopstix [8], a lightweight monitoring tool, relies on a small collection of rules to guide diagnosis in production systems. They describe a recurrent problem at a production system that caused nodes to crash every 1-7 days. Shortly before such crashes they observed that ssh sessions to nodes would stall for tens of seconds. They observed that the symptoms of this problem matched the rule "*if* combined value of CPU utilization for processes is low, and scheduling delay is high *then* kernel bottleneck is likely". This rule led them to trace the problem to a tight loop in kernel's scheduler.

Types of rule-based techniques One approach for representing rules is *codebooks* [26,94] which map each problem to a unique signature consisting of symptoms in both the faulty component where the problem occurs, and related components affected by the original problem. The codebook is instantiated as a dependency matrix where the columns represent the problems, and the rows represent the symptoms. Problems are uniquely diagnosable if all the columns

are different. Codebooks diagnose the underlying problem by identifying the closest match to the observed symptoms.

Other diagnosis tools, such as Chopstix [8] and Vertical Profiling [31] rely on a small collection of rules based on the semantics of the application, and the underlying behavior of the operating system to map changes in system performance on individual nodes to known problems. These tools provide an intuitive approach for diagnosing problems on individual nodes, however they currently do not correlate metrics across multiple nodes and do not address problems that can propagate across the network in distributed systems.

Diagnosis tools that analyze large sets of rules require more sophisticated techniques, such as expert systems that rely on forward inferencing to synthesize results and resolve conflicts when multiple rules fire. These expert systems allow administrators to cope with the deluge of alarms generated by large-scale distributed systems. JECTOR [57] presents a specification language for expressing rules that captures the timing relationship among correlated events. For example, alert operator if a link is down and no corresponding link up event occurs within 2 minutes. Commercial tools such as `HP Operations Manager` [32] use an optimized Rete algorithm [29] to perform pattern matching on rules in a scalable manner that is independent of the number of rules.

Limitations Rule-based approaches are prevalent in commercial tools, such as `IBM Tivoli Enterprise Console` [34] and `HP Operations Manager` [32], as they offer an intuitive approach for expressing system behavior that allows users to augment the rule-base by developing new rules tailored to their unique operating environments. In addition, rule-based systems do not require profound understanding of the underlying system architectural and operational principles. However, rule-based systems suffer from the inability to learn from experience, and the inability to deal with problems not described within the rule-base. Rule-based systems are also difficult to maintain because the rules frequently contain hard-coded network configuration information [86].

2.2 Model-based

Model-based techniques define a mathematical representation of a system, and test the observed state of the system against the learned model to diagnose problems. Some models represent the normal operation of the system, and detect problems whenever the observed system behavior fails to conform to the learned model. Other techniques generate graphical models of how problems propagate through the system [6,41,48,50], and exploit this knowledge to infer the source of the problem. Alternatively, graphical models [40,76] can represent how successes propagate through the system. These graphical models then analyze patterns of probe failures and successes to infer the source of the problem. Lastly, graphical models may represent expected communication patterns within a system and flag problems whenever these patterns are violated.

Illustrative example Sherlock [6] localizes performance problems in large enterprise networks using a graphical model of how errors propagate to infer the source of the problem. Sherlock's inference engine learns service-level dependencies by sniffing packets and detecting which services are likely to be used together, *e.g.*, DNS and web service. Sherlock models three types of components: (i) clients which observe response times delays; (ii) root-cause nodes which are potential sources of faults in the system; and (iii) meta-nodes which model how errors propagate through the system. An example of a meta-node is a fail-over node which requires all nodes in the high-availability group to fail for an error to propagate. If a client observes a high response time, Sherlock uses the fault-propagation model to compute the probability that a client observes a set of symptoms given that a root-cause node is at fault. It outputs a list of root-cause nodes which best explain the observed symptoms at the client.

Types of model-based techniques Model-based techniques can be classified into: (i) *physical model based techniques* which use the physical laws that a system operates under to model constraints on system behavior; (ii) *regression and queuing models* which model relationships between resource consumption and application behavior; and (iii) *graph-theoretic models* which exploit knowledge on how errors or successes propagate in a system to localize problems.

Physical models use models of the physical world, such as the laws of mechanics, electomagnetics, or chemical kinetics to model system behavior and to determine when anomalous behavior is present. They typically model continuous cyber-physical systems in industrial, automotive and aerospace domains whose physics are well understood, *e.g*, powertrain [62] and chassis systems [33] in cars. These systems run in a closed-loop, where sensors monitor the system output, then feed the data into a controller that signals actuators to adjust control as necessary to maintain the desired system output. Problems are diagnosed by executing the physical model alongside the actual system at run-time to detect when the system fails to conform to the model. The fault model typically associated with the control-theoretic approach includes sensor faults, actuator faults, and faults in the mechanical, electromechanical, or hydraulic plant being controlled [52]. Isermann et al. [35] provide a more detailed discussion of these techniques.

Regression and queuing models are useful for workload characterization, capacity planning and detecting performance problems. These models represent relationships between resource consumption and application behavior, and detect anomalies whenever these relationships are violated.

Some techniques model multi-tier Internet applications as queues, and use mean-value analysis [58, 91] to predict transaction response times. These techniques use a network of queues to represent how the tiers in the multi-tier application cooperate to process requests. Mean-value analysis assumes closed queueing models in which the number of clients in the system remains constant. However, it is often difficult in practice to obtain the client session information required to calibrate closed models for real-world production applications [87].

Real-world production workloads are non-stationary, *i.e.*, the relative frequencies of transaction types changes over time. Queuing approaches which leverage regression to learn the relationship between resource consumption and application behavior can be used to predict response times for non-stationary workloads [19, 46, 87]. These models assume that the system contains a small number of types of transactions, and that transaction types strongly influence system resource demands. These models rely on open queues, where clients can join and leave the system model. Open models facilitate more thorough empirical validation in production systems than would be possible with closed models as they do not require client session information [87].

In addition, using queuing theoretic approaches to model transaction mixes allows these systems to distinguish anomalies from workload changes. Cherkasova et al [19] use queues to model the relationship between CPU usage and transaction response times for a transaction mix. They also exploit regression to define an application performance signature that allows them to detect software upgrades by monitoring changes in the application signature. Stewart et al [87] model the relationship between multiple physical resources, namely CPU, disk and network, and response times for a transaction mix. These models need to be re-trained to cope with new transaction types. They also ignore interaction effects across transaction types and implicitly assume that queueing is the only manifestation of congestion.

Graph-theoretic models analyze communication patterns across nodes and processes to model the probability that errors, or successes, propagate through the system. The models may also monitor violations in expected communication patterns. Graph-theoretic models are useful for diagnosing both correctness and performance problems in distributed systems. They can be used to detect multiple independent problems - ranking them by likelihood of occurrence.

SCORE [50] and Shrink [41] localize problems in the IP network by modeling error propagation patterns in the wide-area networks. Both Shrink and SCORE model the system as a two-level graph between the IP layer and the underlying wide-area network. Sherlock [6] and Khanna et al. [48] extend on Shrink and SCORE to deal with multi-level dependencies and with more complex operators that capture load-balancing and failover mechanisms. These techniques infer the root-cause by computing the probability that errors propagate from a set of possible root-cause nodes to the observation nodes. They indict the root-cause nodes that best explain the symptoms at the observation nodes, and scale by assuming that there can only be a small number of concurrent problems in the system at a given time.

Rish et al. [76] propose an active probing approach that exploits a dependency-matrix to represent the failed components that each probe, *e.g.*, server ping, detects. Active probing allows probes to be selected and sent on-demand, in response to one's belief about the state of the system. At each step the most informative next probe is computed and sent. As probe results are received, belief about the system state is updated using probabilistic inference. This process continues until the problem is diagnosed. They extend their active probing approach

to cope with dynamic systems [75], where problems may occur and disappear, by maintaining two sets of probes: one set for repair detection to monitor nodes that are known to have failed, and another set for failure detection to monitor nodes that are known to be working. Their approach assumes a sequential fault model in which only one fault or repair can occur at a time. Joshi et al. [40] use a Bayesian approach to diagnose problems in systems with different types of monitors, or probes, that have differing coverage and specificity characteristics. They use a dependency matrix to represent the probability that a monitor detects a failure in a component, and incrementally update their belief about the set of failed components based on the observed monitor output.

Khanna et al. [47] address diagnosis in distributed systems where errors can propagate across nodes. They track message exchanges between nodes and detect problems by comparing communication patterns against a rule-base of allowed state transitions. Pip [73] detects application-specific problems in distributed systems by allowing programmers to embed expectations about application behavior in the source code. Pip detects problems by comparing actual behavior against expected behavior. Black-box approaches that track message exchanges are more generic and can be easily applied to new systems, whereas white-box approaches like Pip are able to diagnose application-specific problems but require a deeper understanding of system behavior.

Limitations Model-based techniques are well-suited for diagnosing application-specific problems because they encapsulate semantic knowledge on the expected behavior of the system. The incorporation of semantic knowledge can also help them distinguish legitimate changes in behavior, *e.g.* workload changes, from illegitimate changes due to failures [19,46,87]. However, model-based techniques in general require a deep understanding of system behavior to construct the models. Even in cases where automatic model construction is feasible, there is often a tradeoff between the amount of semantic knowledge the model incorporates and the fidelity of the diagnosis. For example, graph-theoretic models [6] that are automatically constructed by examining a system's communication patterns can localize a problem to a single node or a small neighborhood of nodes, but cannot tell what the deeper root cause is. Another disadvantage of model-based techniques is that they can fail to detect novel problems that were not considered in the model.

2.3 Statistical

Statistical techniques for diagnosis summarize and interpret empirical data using techiques such as correlation, histogram comparison and probability theory. These techniques are data-centric and require little expert knowledge or detailed models on system internals. Statistical techniques are either: (i) *parametric* techniques that assume data is drawn from a known distribution, *e.g.*, normal distribution, or (ii) *non-parametric* techniques that do not rely on data belonging to a particular distribution but rather estimate the underlying distribution, *e.g.*,

using histograms or kernel density estimation. Non-parametric methods make fewer assumptions than parametric methods, making them more robust and giving them wider applicability. However, there is a cost - larger sample sizes are required to draw conclusions with the same degree of confidence as parametric methods.

Illustrative example Multivariate Adaptive Statistical Filtering (MASF) [15] is a parametric technique for detecting and visualizing anomalies in data centers. MASF detects anomalies by tracking deviations from the mean in performance counters, such as CPU and memory usage. MASF assumes that data is drawn from a normal distribution and flags an anomaly if a metric exceeds 3 standard deviations from the mean. To cater for seasonal variations in behavior, such as heavy load during the day and light load at night, MASF maintains separate behavioral profiles for computing the mean and standard deviation of each metric. MASF alerts operators to suspicious behavior and allows them to visualize anomalies, but it does not automatically localize the problem.

Types of statistical techniques Statistical techniques are pervasive in problem diagnosis literature. Some model-based techniques discussed earlier rely on statistical techniques, such as correlation and regression, in conjunction with deep knowledge of the application's behavior to diagnose problems. In contrast, the statistical techniques discussed in this section make fewer assumptions about the application's behavior. Statistical techniques can be classified as parametric or non-parametric techniques.

Parametric techniques assume that data is drawn from a known distribution. Normal distributions are commonly used for anomaly detection and diagnosis because of their tractablity, and because normality can sometimes be justified by the *central-limit theorem* which explains why many distributions tend to be close to the normal distribution. These techniques typically detect anomalous behavior by identifying significant deviations from the mean for performance counters, which they assume follow a normal distribution. However, hardware failure rates are better modeled using Weibull distributions which capture the increased failure rates of devices as they age [78, 79].

Agarwal et al [1] use change-point detection and problem signatures to detect performance problems in enterprise systems. They detect abrupt changes in system behavior by monitoring changes to the mean value of performance counters over consecutive windows of time. This technique does not scale well if the number of nodes and metrics is large. NetMedic [42] diagnoses propagating problems in enterprise systems by analyzing dependencies between nodes, and correlations in state perturbations across processes to localize problems. NetMedic represents state for each system component as a vector that indicates whether each metric was anomalous or normal by assuming that each metric obeys a normal distribution and flagging anomalies based on deviation from the mean. If two components which depend on each other are anomalous, NetMedic

searches for time periods where the source component's state is similar to its current state, and searches for destination states that have experienced significant changes in the same period. These destination states are the likely culprits.

Draco [45] performs statistical diagnosis of problems in large Voice-over-IP (VoIP) systems by comparing differences in the distributions of attributes, such as hostnames and customer IP addresses, in successful and failed calls. Draco assumes that these attributes are drawn from a Beta distribution and localizes problems by identifying attributes that are most correlated with failed calls. By comparing successes and failures over the same window of time, Draco avoids the need for separate learning passes, and can thus diagnose problems that have never been seen before.

Non-parametric techniques assume that data is drawn from an unknown distribution. Non-parametric techniques estimate the underlying data distribution using histograms or kernel density estimators, or make generalizations about the populations from which the samples were drawn, *e.g.*, using correlation.

Histogram-based techniques typically diagnose problems by comparing histograms of performance counters before and during an anomalous period to identify the metrics most likely to be associated with the problem. Tan et al. [68,89] diagnose problems in large clusters using histogram-comparison of performance counters to identify "odd-man-out" behavior. Peer-comparison allows their approach to be robust to workload changes. However, propagating errors, *e.g.*, packet-loss that affects communication across multiple nodes, reduces the accuracy of their approach. Shen et al. [81] propose a reference-driven approach to diagnose performance problems due to configuration changes or upgrades. Their approach relies on histogram comparison to identify the collection of single-parameter changes that best explain the performance deviation observed.

Correlation-based techniques analyze historical data to automatically discover relationships between pairs of metrics that are stable over time [38, 39]. Changes in these learned correlations may signal problems. Correlation can also be used to automatically discover causal relationships between metrics in distributed systems. Giza [60] exploits knowledge of the system's topology to identify spatial correlations between events. For example, to detect that customers in Texas are experiencing poor video quality. Next, Giza uses cross correlation to discover causal relationships between the observed symptoms and root-cause events. Oliner et al. [67] also use cross correlation to discover causal relationships between anomaly signals across components. The anomaly signals represent the changes in the behavior of components over time in terms of resource usage, message timing or semantics. Project5 [4] records packet traces at each node and uses message correlation algorithms to automatically extract end-to-end causal traces for requests, and detect high-latency paths. Correlation-based approaches can discover spurious relationships depending on the thresholds used to determine whether a correlation is significant. In addition, correlation-based approaches do not scale well if the number of nodes and metrics is large because they search for metric correlations both locally, and remotely between nodes communicating with each other.

Dimensionality-reduction techniques, *e.g.*, Principal Component Analysis, can reduce the number of metrics to compare when diagnosing problems by summarizing dominant trends. Xu et al [93] use source-code analysis to apply structure to console logs and discover dominant historical trends in application state and message counts using Principal Component Analysis. PeerWatch [43] uses peer-comparison to detect anomalies in heterogeneous clusters running different hardware. Their peer-comparison algorithm uses a dimensionality-reduction technique known as canonical correlation analysis to normalize performance differences due to different hardware, and discover correlations between peers.

Limitations Statistical techniques require little expert knowledge or detailed models of system internals. The diagnosis techniques can rely on well-established statistical theories to ground their algorithms, and test that their results are statistically significant, *i.e.*, unlikely to have occurred by chance alone. For example, hypothesis tests such as the t-test, allow us to reject the hypothesis that the observed system behavior is consistent with the expected system behavior with a degree of confidence. When building statistical profiles of behavior, care must be taken to include sufficient data samples and test assumptions on data distributions to ensure validity. For example, incorrectly assuming that the data is drawn from a normal distribution can lead to a high error rate. Since statistical techniques do not incorporate much semantic knowledge about semantic behavior, they can experience difficulties distinguishing legitimate changes in behavior such as workload changes from performance problems.

2.4 Machine Learning

Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to evolve behaviors based on training data. Machine-learning techniques borrow heavily from statistical techniques, *e.g.*, data distributions and probability theory. Machine learning relies on training and cross-validation which involves partitioning a sample of data into complementary subsets, performing the analysis on one subset called the training set, and validating the analysis on the other subset called the validation set or testing set. Cross-validation can provide an estimate of model accuracy.

Illustrative example Cohen et al. [21] describe an approach for automatically extracting signatures of system behavior so that operators can identify and quantify recurrent problems, *e.g.*, slowdowns due to insufficient database connections. They use Service-Level Objective (SLO) violations to identify periods of time where the system was behaving abnormally and use tree augmented Bayesian networks (TANs) to determine which metrics are most correlated with the anomalous periods. They build signatures of the anomalous periods using metric attribution as follows: *1* indicates a metric is selected by model and attributed to failure, *-1* indicates a metric is selected by model but not attributed to failure, and *0* indicates a metric was not selected by model (irrelevant). They

cluster the signatures based on a purity score which indicates what fraction of signatures in the cluster are associated with failures. Clusters with greater purity provide more confidence in the signature. They found that the metric attribution gives better results than using raw metric values. They also found that they can leverage signatures from different sites to identify or rule out recurrent problems.

Types of machine learning techniques Diagnosis algorithms that rely on machine learning can be categorized into two broad categories namely: (i) unsupervised learning which identifies patterns in unlabeled data typically through clustering, and (ii) supervised learning which infer a function that best classifies successful and failed states from labeled data.

Unsupervised learning identifies patterns in unlabeled data typically through clustering, and detects unexpected outlier data points that might be indicators of failures.

Kiciman and Fox [49] uses probabilistic context-free grammars to model the causal paths in the system. The grammar rules represent the probability that one component calls another. They identify anomalous causal paths by measuring the difference between the probability of the observed transition and the expected probability of the transitions that make up the causal path. Magpie [7] uses a string-edit-distance comparison to group together requests with similar behaviour, from the perspective of request structure, synchronization points and resource consumption. The representative requests from each clusters allow them to construct concise workload models and detect outlier requests.

Supervised learning uses labeled data of successful and failed states to learn which metrics are most correlated with failed states, or to identify signatures of recurrent problems from a database of known problems.

Metric attribution approaches localize problems by identifying resource-usage metrics or components that are highly correlated with failed states. They allow operators to sift through the hundreds or thousands of metrics available in their system and narrow down the handful of metrics that yield insight to the cause of the problem and its location and guide operators in performing more detailed root-cause analysis. Once the operators determine the root-cause, they can then annotate the output of metric attribution with the root-cause and build the database of known problems used by signature-based approaches.

Pinpoint [17] and MinEntropy [18] localize components highly correlated with failed requests using data clustering [17] or decision trees [18]. They represent requests using a matrix where each row is a client request, and columns are components. An additional column indicates whether the request was successful or failed. The matrix serves as input into the machine learning algorithm. These approaches detect problems that result in changes in the causal flow of requests such as exceptions. More recently, Spectroscope [77] categorizes requests based on functionality, *e.g.*, read or write requests, and applies data clustering to requests in each category to identify outliers due to changes in causal flows or request durations. Some limitations of these approaches are that they cannot

distinguish between sets of components that are tightly coupled and are always used together, and they require requests to be independent of each other. If a request corrupts state and affects subsequent requests, the non-independence of requests makes it difficult to detect the real faults because the subsequent requests may fail while using a different set of components [17].

Cohen et al. [20] use tree augmented Bayesian networks to determine which resource-usage metrics are most correlated with the anomalous periods. They proposed an extension [96] to their work that uses ensembles of Bayesian models to adapt to changing workloads and infrastructure.

Signature-based approaches allow system administrators to identify recurrent problems from a database of known problems. Signature-based approaches have wide applicability because studies have shown that typically half, and as much as 90% of software failures are due to recurrent problems [25]. Research has centered on how to represent and retrieve signatures of known problems from the database of known problems. However, these approaches do not fare well at automatically identifying problems that have not previously been diagnosed.

Yuan et al. [95] learn signatures of known problems in standalone systems by analyzing sequences of system calls. They target problems that have the same manifestation, *e.g.*, a web page may fail to load due to different underlying root causes such as an invalid IP address or an unplugged network cable. Analyzing system calls allows them to distinguish between problems that might be indistinguishable when analyzing resource usage data. They use multi-class Support Vector Machines to learn signatures of problems. However, their approach does not address distributed systems.

Cohen et al. [21] and Bodik et al. [9] generate signatures of recurrent problems in distributed systems by using the discrete feature vectors obtained through metric attribution. They found that using discrete values to represent signatures performs better than using real-valued metrics. In addition, they found that they can leverage signatures learned at one geographical location to diagnose problems in data centers at a different location.

Duan et al [25] present an approach that can be used for both known problems, and problems that have not previously been seen. They use a supervised approach (decision trees or signature databases) to identify recurrent problems. If the current failure does not match the annotated failures in the database, they compare it to the healthy data to identify features that are correlated with the failure. They then select multiple instances of the same failure which they can present to the system administrator to annotate.

Limitations Machine-learning techniques automatically learn profiles of system behavior, for example, using clustering to identify signatures of known problems. Machine-learning can also help localize problems by identifying resource-usage metrics or components that are highly correlated with failed states. However, these techniques can suffer from the curse of dimensionality that reduces accuracy when the number of features is large. Additionally, they are also susceptible to *overfitting*, a phenomenon in which the learner learns features of the evidence

that are circumstantial rather than those that actually define the relationship between the faults and their effects. Over-fitted models generalize poorly, and can fail when presented with evidence that is only slightly different from the one on which the model was trained. Finally, because machine learning techniques learn a direct mapping between the symptoms and underlying root causes without an intermediate structural model of the system, lengthy retraining is required whenever the system behavior changes significantly. Furthermore, previously learned models often have to be thrown away during the period of retraining, leaving the system vulnerable to any problems. Therefore, machine learning techniques may not be appropriate for systems that are upgraded frequently.

2.5 Visualization

Automated diagnosis tools might not always be available, and when available they occasionally miss the true root-cause and typically reduce the search space to a small number of likely culprits [59]. Visualization tools allows operators to cope with these scenarios by: (i) summarizing data trends, (ii) supporting interactive graphs that allow operators to explore different hypotheses on the root-cause of problems, and (iii) integrating output from automated diagnosis tools.

Visualization tools [30, 83, 85] provide an array of simple graphs, *e.g.* line plots, barcharts, and histograms, to display trends in performance counters such as CPU utilization. They use simple statistical tests such as the deviation from the mean to flag outliers, and use color to highlight these outliers. LiveRAC [63] is a visualization system that supports the analysis of large collections of system management timeseries data consisting of hundreds of parameters across thousands of network devices. LiveRAC provides high information density using a reorderable matrix of charts, with semantic zooming that dynamically adapts different aspects of each chart based on available space.

Magpie [7], X-trace [28], and Dapper [83] are primarily tools for tracing causal request paths, but they also offer support for visualizing requests whose causal structure or duration is anomalous. Artemis [23] provides a pluggable framework for distributed log collection, data analysis, and visualization. Mochi [90] is a log-analysis based debugging tool that visualizes both the flow of data and the flow of control for a large-scale parallel processing framework known as Hadoop. NetClinic [59] visualizes data from computer networks using directed graphs, and presents suggested diagnostics for observed problems by incorporating output from an automated analytic reasoning engine [42].

2.6 Count-and-threshold techniques

Physical faults are distinguished by their nature and duration of impact as being permanent or temporary [5]. Permanent faults may lead to error whenever the component is activated; the only way to handle such faults is to remove the affected component. Temporary faults can be internal (usually known as intermittent) or external (transient). The former are caused by some internal part

deviating from its specified behavior. After their first appearance, they usually exhibit a relatively high occurrence rate and, eventually, tend to become permanent. On the other hand, transient faults, often manifesting the encountered interferences as noise-pulses on the communication channels, cannot be easily traced to a defect in a particular part of the system and, normally, their adverse effects tend to disappear. In industries like transportation and telecommunications, where operating with permanently faulty modules would carry high risks or costs, it is common that modules, disconnected because they were considered faulty, are later proved to be free from permanent faults when tested during repair operations. Therefore, treating transient faults as permanent has a high cost for these industries. A good discrimination between transient and intermittent/permanent faults solves two important problems: i) prevents the undue removal of nodes affected by transient faults, thus avoiding unnecessary depletion of system resources; and ii) helps to maintain the correct coverage of the system fault hypotheses (*i.e.*, the assumption on the number of faults tolerated by the core system protocols within a given time window) by keeping in operation nodes not permanently faulty. Considering that most perturbations encountered are transient [22, 82], the issue of *proper* diagnosis of transients is a significant issue of interest.

Illustrative example A generic class of online low-overhead count-and-threshold mechanisms, called alpha-count, has been initially proposed in [11] and later enriched with a double threshold in [12]. It is characterized by: a) tunability through internal parameters, to warrant wide adaptability to a variety of system requirements; b) generality with respect to the system in which they are intended to operate, to ensure wide applicability; c) simplicity of operation to allow high analyzability through analytical models and to be implementable as small, low-overhead and low-cost modules, suitable especially for embedded, real-time, dependable systems. In its basic formulation, an error counter is associated to each component, which is incremented when the component fails and decremented when it delivers a correct service. When the value of the counter exceeds a given threshold value , the component is diagnosed as affected by a permanent or an intermittent fault.

Heuristic mechanisms The importance of distinguishing transient faults, so that they can be dealt with specifically, is testified by the wide range of solutions proposed, although with reference to specific systems (*e.g.*, [3, 36, 55, 65, 84], just to mention a few). Most of these solutions are based on more or less simple heuristic mechanisms. One commonly used method, for example, in several IBM mainframes [84], is to count the number of error events: too many events in a given time frame would signal that the component needs to be removed. In TMR MODIAC, the architecture proposed in [65], two failures experienced in two consecutive operating cycles by the same hardware component that is part of a redundant structure make the other redundant components consider it as definitively faulty. Another architecture using similar mechanisms, designed for distributed ultra-dependable control systems, is described in [51]. In this case, a

combination of diversified design, temporal redundancy and comparison schema is used to obtain a detailed determination of the nature of faults. Counting mechanisms are also used to solve the so called 2-2 splits, *i.e.*, to determine the correct value among four proposals in a quadruple modular redundancy (QMR) system when there is a tie. In [3], a list of *suspect* processors is generated during the redundant executions; a few schemes are then suggested for processing this list, *e.g.*, assigning weights to processors that participate in the execution of a job and fail to produce a matching result and taking down for diagnostics those whose weight exceeds a certain threshold. Other approaches do, instead, concentrate on off-line analysis of system error logs, and therefore are not applicable on-line. In [55], some heuristics, collectively named Dispersion Frame Technique, for fault diagnosis and failure prediction are developed and applied to system error logs taken from a large Unix-based file system. The heuristics are based on the inter-arrival patterns of the failures (which may be time-varying). For example, there is the 2-in-1 rule, which warns when the time of inter-arrival of two failures is less than one hour, and the 4-in-1 rule, which fires when four failures occur within a 24-hour period. In [36], an error rate is used to build up error groups and simple probabilistic techniques are then recursively applied to discern similarities (correlations) which may point to common causes (permanent faults) of a possibly large set of errors.

Other count-and-threshold solutions In [10], a methodology and an architectural framework for handling multiple classes of faults (namely, hardware-induced software errors in the application, process and/or host crashes or hangs, and errors in the persistent system stable storage) in a COTS and legacy-based application have been defined. Also, a generic FDIR (Fault Detection followed by Isolation and system Reconfiguration) framework for integrating existing distributed diagnosis approaches with a count-and-threshold algorithm is proposed in [80].

Formulation based on Bayesian inference Another direction of research has addressed a rigorous mathematical formulation of diagnosis based on Bayesian inference [71]. Bayesian inference provides a standard procedure for an observer who needs to update the probability of a conjecture on the basis of new observations. Therefore, after a new observation is provided by the error detection subsystem, the on-line diagnosis procedure produces an updated probability of the module being affected by a permanent fault. This leads to an optimal diagnosis algorithm, in the sense that fault treatment decisions based on its results would yield the best utility among all alternative decision algorithms using the same information. This higher accuracy with respect to simple heuristics comes at the cost of higher computational complexity.

Formulation based on Hidden Markov Models A formalization of the diagnosis process, addressing the whole chain constituted by the monitored component, the deviation detector and the state diagnosis through Hidden Markov Models has been proposed in [24], with the goal of developing high accuracy diagnosis processes based on probabilistic information rather than on merely

intuitive criteria-driven heuristics. Because of its high generality and accuracy, the proposed approach could be usefully employed: i) to evaluate the accuracy of low-cost on-line processes to be adopted as appropriate and effective diagnostic means in real system applications; ii) for those diagnostic mechanisms equipped with internal tunable parameters, to assist the choice of the most appropriate parameter setting to enhance effectiveness of diagnosis; and iii) to allow direct comparison of alternative solutions.

Limitations The accuracy of diagnosis performed through threshold-based mechanisms strongly depends on proper calibration of the mechanism parameters, namely the threshold's value and the function adopted to update the counter. Actually, proper setting of the mechanism's parameters is fundamental to trade between accuracy and promptness, which are the typical contrasting requirements to be satisfied in fault discrimination, that is:

- To signal, as quickly as possible, all components affected by permanent or intermittent faults. Gathering information to discriminate between transient and intermittent faults takes time, thus giving rise to a longer fault latency. This increases the chances of catastrophic failure and also increases the requirements on the error processing subsystem in fault tolerant systems.
- To avoid signaling components that are not affected by permanent or intermittent faults. In fact, depriving the system of resources that can still do valuable work may be even worse than relying on a faulty component.

Practitioners have long used expertise and trial-and-error approach to tune their systems. However, solutions based on rigorous mathematical formulations, such as alpha-count and its variants, are amenable to high analyzability of the parameters tuning through analytical models. Therefore, the system designer is equipped with a systematic, predictable, and repeatable way to identify a proper setting, taking into account requirements of the targeted application field.

3 Industrial Applications

In this section we summarize the use of the previously described diagnosis techniques in several industrial applications ranging from large scale telecommunications infrastructures, to Internet services, to embedded systems and the automotive industry, to aerospace and unmanned spacecraft.

3.1 Telecommunications

The telecommunications industry has long operated some of the largest scale distributed systems in use - from digitally switched phone networks and Internet backbones, to high-speed cellular networks and Internet Protocol Television (IPTV) deployments. The high resilience requirements of these systems have led to widespread deployment of diagnosis techniques by telecom operators. [86] provides a survey of diagnosis techniques for communication systems.

Work in the telecom domain has traditionally revolved around alarms produced by network elements, and trouble ticket systems [54] that track and coordinate troubleshooting. The use of rule-based expert systems for troubleshooting was common - as early as 1990, Wright et al. [92] survey a list of 40 rule-based expert system in use within the telecom industry. More recently, codebook-based approaches [26, 94] have been used to correlate alarms across many network devices to a single "root cause" alarm that the operators can investigate. SCORE [50] uses a model of how IP links are routed over an underlying optical network to localize optical layer failures (*e.g.*, fiber-cuts) based on IP layer loss measurements. *rcc* [27] uses static analysis to detect faults in BGP router configurations by checking them against a high-level correctness specification.

However, such knowledge-based techniques often fail to capture emergent behaviors that are rife in highly heterogenous telecom networks. Therefore, increasing attention is being devoted to "knowledge-free" techniques such as statistical methods and machine learning. Giza [60] uses spatial (*i.e.*, in the same geographical neighborhood) and temporal correlations between network alarms in a large IPTV network to determine the true root cause of network outages that result in many alarms across different layers (e.g., video, TCP, IP). Draco [45] performs statistical comparisons between successful and dropped calls in a large voice-over-IP (VoIP) service to identify features that discriminate the failures. Mahimkar et al. [61] perform statistical comparisons of various performance metrics such as CPU utilization and loss of network elements before and after upgrades to identify problems that result from upgrades.

3.2 Internet Services and Data Centers

Diagnosis in data center applications has centered on interactive applications in Internet Services [17, 21, 49, 77] and enterprise systems [6, 42, 95], and batch applications in data-intensive computing [68, 89, 93]. Interactive applications typically have well-established Service Level Objectives, *e.g.*, 99% of Internet requests should be serviced within 4 seconds, to ensure high-availability. Some techniques use metric attribution [9, 17, 21, 77] localize problems by identifying resource-usage metrics or components that are highly correlated with failed states. Signature-based techniques [9, 21, 25, 95] have been used to diagnose recurrent problems by generating signatures of known problems using techniques such as metric attribution. Regression and queuing models [19, 46, 87] detect performance problems in Internet services by modeling the relationships between performance counters, *e.g*, CPU utilization and application response times, and detecting performance anomalies whenever these relationships are violated.

Batch applications in data-intensive computing have more diverse runtimes [44]. Peer-comparison techniques [64, 68, 89] diagnose problems by exploiting the parallelism inherent in these applications to compare behavior across components and detect "odd-man-out" behavior. The distributed nature of data center applications facilitates the use of graphical models to analyze communication patterns across nodes (or processes) to model the probability that errors [48],

or successes [76] propagate through the system. Log analysis [66, 67, 93] and rule-based techniques [8, 31, 32] are also widely used in data center applications.

3.3 Embedded Systems

Embedded systems are computer systems designed to do one or a few dedicated functions, often with real-time computing constraints. Embedded systems are present in a large variety of systems such as consumer electronics (*e.g.*, mobile phones), and automotive safety-critical systems (*e.g.*, anti-lock braking, and drive-by-wire systems). Lanigan et al. [52] provides a comprehensive survey of failure diagnosis in automative systems.

Preparata et al. [72] proposed the Preparata, Metze, and Chien (PMC) model to identify faulty components by collating results of diagnostic tests across a distributed system. Heuristic mechanisms based on thresholds have been also adopted, such as [65] in railway control systems. Serafini et al. [80] distinguish between healthy nodes from unhealthy nodes in time-triggered automotive systems by applying penalties and rewards to the collated diagnostic tests. The penalty counter is increased when a node's entry in the consistent health vector indicates a fault, otherwise the reward counter is increased according to the criticality of the node. When the reward threshold for a node is crossed, the penalty counter for that node is reset to zero. When the penalty threshold for a node is crossed, the node is diagnosed as faulty. Peti et al. [70] introduce Out-of-norm Assertions (ONAs) as a way to correlate fault effects in the three dimensions of value, time and space. They use ONAs to describe fault patterns that discriminate between different types of faults, *i.e.*, wearouts, massive transient faults, and connector faults in automotive systems. Other diagnosis techniques for embedded systems rely on physical models to diagnose problems in powertrain [62] and chassis systems such as braking [33] in cars.

3.4 Aerospace

Stroupe et al. [88] and Patton [69] provide a detailed survey on diagnosis in aerospace systems. Livingstone is a model-based system, developed at NASA Ames, used to autonomously control the New Millennium Deep Space One Probe (DS 1) [88]. Livingstone accepts a model of the components of a complex system such as a spacecraft or chemical plant and infers from it the overall behavior of the system. From this, Livingstone monitors the operation of the system, diagnoses its current state, determines if sensor readings are implausible, and recommends actions to put the system into a desired state even in the face of failures. MARPLE is an expert system that relies on a model-based technique known as constraint suspension to diagnose problems [88]. Constraint suspension views the system to be monitored as a network of black-box components and places constraints on the behavior of each component. When observed behavior violates these constraints, MARPLE suspends the components in the network, one at a time until it finds a component that can account for all the inconsistent

values at the nodes. MARPLE has been demonstrated to work for the NASA LRC Space Station Freedom (SSF) power system testbed.

Kalman filtering [14] is a state and parameter estimation technique that fuses data from different sensors together to produce an accurate estimate of the true system state. Jayakumar and Das [37] use a single Kalman filter, driven by the motor shaft velocity sensor, to diagnose problems in a flight control system. They diagnose incipient sensor faults using structured residuals that are generated using the Kalman filter estimates. Patton [69] discusses the use of filters to diagnose faults in flight control systems. At the moment, the analytic redundancy provided by model-based approaches cannot be used to replace hardware redundancy due to the safety-critical nature of aerospace applications. However, analytical redundancy can be used to suppress some levels of replication, *e.g.*, to replace quadruple by triplex schemes [69].

4 Future Trends and Challenges

Despite the tremendous progress that has been made in automated fault diagnosis, many open problems remain. Below, we enumerate a few such problems that may serve to inspire new contributions in the field.

4.1 Online Recovery and Self Healing

The eventual outcome of any automated diagnosis technique is the identification and removal of any impairments to a system's proper operation. Therefore, a natural evolution of diagnosis is the construction of "self-healing" systems that can automatically perform recovery actions upon the outcome of an online diagnosis procedure to remove faults. Self-healing is relatively risk-free either when the fault detection and diagnosis mechanisms are highly accurate, or when the recovery actions do not impose any penalties if applied wrongly. For example, JAGR [16] presents an autonomous self-recovering Enterprise Java Bean (EJB) application server that allows recovery using quick microreboots of components. The basic philosophy in that work is to make recovery mechanisms cheap enough that they can be liberally applied without consequences even if diagnosis produces poor outcomes. When recovery actions are not cheap, self-healing becomes a risky proposition because wrong diagnosis can lead to poor recovery decisions.

[40] propose a decision theoretic framework using Partially Observable Markov Decision Processes (POMDPs) to reason about recovery decisions of different costs under uncertain fault diagnoses. They combine the decision algorithm with a graph-theoretic diagnosis algorithm to determine when components of a multi-tier Enterprise system should be rebooted using the results of end-to-end system tests. [56] propose a model-free approach for choosing recovery actions by using reinforcement learning to learn the effectiveness of previously executed actions as a function of the observable symptoms. However, none of these techniques are sufficient when faced with unanticipated problems due to emergent behavior.

4.2 Automatic Model Construction

Although model-based techniques have several advantages such as the ability to predict error propagation, the ability to provide semantically meaningful diagnoses, and the ability to cope with structural system changes without the need to relearn, they require detailed and accurate models that have to be constantly updated. There is some literature on automatically constructing system models, primarily those suitable for graph-theoretic approaches, but also some on learning queuing-theoretic models. Examples include work on automatic determination of component dependencies by system perturbation (e.g., [13]), work on dependency generation via passive observation (e.g., [4], [74], [2]), approaches based on statistical clustering (e.g., [18]), and approaches to learn the parameters of queuing models using statistical regression [87]. However, all of these techniques are only suitable for learning models of a system during normal operations. Learning the dependencies of a system that may be exercised during fault modes is an open problem whose solution is likely to require a combination of static analysis (to discover all dependencies) along with runtime measurement (to identify those dependencies which are explained by normal behaviors).

4.3 Cross Domain and Cross Layer Diagnosis

In many domains such as internet services and telecommunications, large systems are increasingly built as a composition of multiple horizontal "technology layers" and vertical "administrative domains". For example, consider a typical internet application constructed using the Java runtime and its libraries, hosted in a Tomcat application server running on a Linux OS inside a virtual machine that runs on a Windows host running in a rack in a particular data center of a cloud provider. For communication with a backend database, it uses the Simple Object Access Protocol (SOAP) that runs over HTTPS (secure HTTP) that runs over an IP virtual private network that is provisioned over an Ethernet service provided by an Internet Service Provider (ISP) that provisions it as a tunnel over an MPLS (multi-protocol label-switching) backbone network. In addition, this application uses the Bing mapping service from Microsoft, obtains analytics support from Google Analytics, and uses PayPal as a payment service. Each of these services also run on very similar infrastructure layers, and depending on which cloud provider the application users, some of these services may also share a data-center and/or network provider with the application.

In such a highly layered and highly silo'ed setup, faults can occur in each of the technology layers or third party providers the service uses. Furthermore, symptoms of lower layer problems (e.g., packet loss on the MPLS network) can translate into symptoms in higher layers (slow response from database server). Seemingly independent third party providers may have common dependencies - e.g., they use the same cloud provider, resulting in correlated failures. No single layer or administrative domain may have sufficient information to completely determine the root cause of a fault occurring in the system. These complications make diagnosis a challenging task. Although there has been some preliminary

work on combining information across technology layers (e.g., [50, 60]), comprehensive approaches that can take a whole system view when performing diagnosis are still elusive.

5 Conclusions

Diagnosis of failures occurring in systems in the field is an important aspect of system resilience and its assessment. In this chapter, we provided a broad overview of automated techniques for fault diagnosis ranging from knowledge-based techniques that encode expert knowledge in the form of rules or system models to model-free techniques that rely on statistical correlations, regression, and machine learning to perform some aspects of the diagnosis task without any prior human knowledge. We provided examples of industrial applications in which automated diagnosis has proven to be a valuable tool for ensuring and evaluating resilience. Today, while these mainly include telecommunications and Internet services that have to deal with issues of scale and automotive and aerospace systems that have to deal with the absence of human expertise when problems occur, automated diagnosis is poised to make a foray into an increasingly number of domains ranging from software debugging tools to agents that help troubleshoot configuration problems in personal computer systems. Finally, we review some of the open problems in this area - these include the need to deal with problems that occur due to emergent, unpredictable behaviors, and the need for recovery techniques to automatically act upon the output of diagnosis algorithms.

References

1. M. K. Agarwal, M. Gupta, V. Mann, N. Sachindran, N. Anerousis, and L. B. Mummert. Problem determination in enterprise middleware systems using change point correlation of time series data. In *IEEE/IFIP Network Operations and Management Symposium*, pages 471–482, Vancouver, Canada, April 2006.
2. S. Agarwala, F. Alegre, K. Schwan, and J. Mehalingham. E2eprof: Automated end-to-end performance management for enterprise systems. In *IEEE Conference on Dependable Systems and Networks*, pages 749–758, June 2007.
3. P. Agrawal. Fault tolerance in multiprocessor systems without dedicated redundancy. *IEEE Transanctions on Computers*, 37:358–362, 1988.
4. M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed system of black boxes. In *ACM Symposium on Operating Systems Principles*, pages 74–89, Bolton Landing, NY, October 2003.
5. A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.
6. P. Bahl, R. Chandra, A. G. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 13–24, Kyoto, Japan, August 2007.

7. P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 259–272, San Francisco, CA, December 2004.
8. S. Bhatia, A. Kumar, M. E. Fiuczynski, and L. L. Peterson. Lightweight, high-resolution monitoring for troubleshooting production systems. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 103–116, San Diego, CA, December 2008.
9. P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen. Fingerprinting the datacenter: automated classification of performance crises. In *European conference on Computer systems (EuroSys)*, pages 111–124, Paris, France, April 2010.
10. A. Bondavalli, S. Chiaradonna, D. Cotroneo, and L. Romano. Effective fault treatment for improving the dependability of cots and legacy-based applications. *IEEE Transactions on Dependable and Secure Computing*, 1:223–237, 2004.
11. A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni. Discriminating fault rate and persistency to improve fault treatment. In *IEEE FTCS International Symposium on Fault-Tolerant Computing*, pages 354–362, 1997.
12. A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni. Threshold-based mechanisms to discriminate transient from intermittent faults. *IEEE Transactions on Computers*, 49:230–245, 2000.
13. A. Brown, G. Kar, and A. Keller. An active approach to characterizing dynamic dependencies for problem determinination in a distributed environment. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 377–390, Seattle, WA, May 2001.
14. R. G. Brown and P. Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. John Wiley and Sons, USA, 1997.
15. J. P. Buzen and A. W. Shum. MASF – multivariate adaptive statistical filtering. In *International Computer Measurement Group Conference*, pages 1–10, Nashville, TN, December 1995.
16. G. Candea, E. Kiciman, S. Zhang, P. Keyani, and A. Fox. JAGR: An autonomous self-recovering application server. In *Autonomic Computing Workshop*, pages 168–177, 25 June 2003.
17. M. Chen, E. Kiciman, E. Fratkin, E. Brewer, and A. Fox. Pinpoint: Problem determination in large, dynamic, internet services. In *IEEE Conference on Dependable Systems and Networks*, pages 595–604, Bethesda, MD, June 2002.
18. M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *IEEE International Conference on Automatic Computing*, pages 36–43, New York, NY, May 2004.
19. L. Cherkasova, K. M. Ozonat, N. Mi, J. Symons, and E. Smirni. Anomaly? application change? or workload change? Towards automated detection of application performance anomaly and change. In *IEEE Conference on Dependable Systems and Networks*, pages 452–461, Anchorage, Alaska, June 2008.
20. 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 *USENIX Symposium on Operating Systems Design and Implementation*, pages 231–244, San Francisco, CA, December 2004.
21. I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *ACM Symposium on Operating Systems Principles*, pages 105–118, Brighton, United Kingdom, Oct 2005.
22. C. Constantinescu. Trends and challenges in VLSI circuit reliability. *IEEE Micro*, 23:14–19, 2003.

23. G. F. Cretu-Ciocarlie, M. Budiu, and M. Goldszmidt. Hunting for problems with Artemis. In *USENIX Workshop on Analysis of System Logs*, San Diego, CA, December 2008.
24. A. Daidone, F. Di Giandomenico, A. Bondavalli, and S. Chiaradonna. Hidden Markov models as a support for diagnosis: Formalization of the problem and synthesis of the solution. In *IEEE Symposium on Reliable Distributed Systems*, pages 245–256, Leeds, UK, October 2006.
25. S. Duan and S. Babu. Guided problem diagnosis through active learning. In *IEEE International Conference on Automatic Computing*, pages 45–54, Chicago, IL, June 2008.
26. EMC. Automating root cause analysis: EMC Ionix codebook correlation technology vs. rule-based analysis. Technical Report h5964, EMC, November 2009.
27. N. Feamster and H. Balakrishnan. Detecting BGP configuration faults with static analysis. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 43–56, Boston, MA, May 2005.
28. R. Fonseca, G. Porter, R. Katz, S. Shenker, and I. Stoica. X-Trace: A pervasive network tracing framework. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 271–284, Cambridge, MA, April 2007.
29. C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(4):17–37, 1982.
30. Ganglia. Ganglia monitoring system, 2007. http://ganglia.info.
31. M. Hauswirth, A. Diwan, P. Sweeney, and M. Hind. Vertical profiling: Understanding the behavior of object-oriented applications. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 251 – 269, Vancouver, BC, Canada, October 2004.
32. Hewlett Packard. HP operations manager, 2010. http://www.managementsoftware.hp.com.
33. K. Huh, K. Han, D. Hong, J. Kim, H. Kang, and P. Yoon. A model-based fault diagnosis system for electro-hydraulic brake. SAE Technical Paper Series 2008-01-1225, SAE International, Warrendale, PA, USA, April 2008.
34. IBM. Tivoli Enterprise Console, 2010. http://www.ibm.com/software/tivoli/products/enterprise-console.
35. R. Isermann. Model-based fault-detection and diagnosis - status and applications. *Annual Reviews in Control*, 29(1):71–85, 2005.
36. R. K. Iyer, L. T. Young, and P. V. K. Iyer. Automatic recognition of intermittent failures: An experimental study of field data. *IEEE Transactions on Computers*, 39:525–537, 1990.
37. M. Jayakumar and B. Das. Diagnosis of incipient sensor faults in a flight control actuation system. In *SICE-ICASE, 2006. International Joint Conference*, pages 3423 –3428, Busan, Korea, October 2006.
38. G. Jiang, H. Chen, K. Yoshihira, and A. Saxena. Ranking the importance of alerts for problem determination in large computer systems. In *IEEE International Conference on Automatic Computing*, pages 3–12, Barcelona, Spain, June 2009.
39. M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. S. Ward. System monitoring with metric-correlation models: problems and solutions. In *IEEE International Conference on Automatic Computing*, pages 13–22, Barcelona, Spain, June 2009.
40. K. R. Joshi, W. H. Sanders, M. A. Hiltunen, and R. D. Schlichting. Automatic model-driven recovery in distributed systems. In *IEEE Symposium on Reliable Distributed Systems*, pages 25–38, Orlando, Florida, October 2005.

41. S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A Tool for Failure Diagnosis in IP Networks. In *ACM SIGCOMM Workshop on mining network data (MineNet-05)*, Philadelphia, PA, August 2005.
42. S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 243–254, Barcelona, Spain, August 2009.
43. H. Kang, H. Chen, and G. Jiang. PeerWatch: a fault detection and diagnosis tool for virtualized consolidation systems. In *IEEE International Conference on Automatic Computing*, pages 119–128, Washington, DC, June 2010.
44. S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production MapReduce cluster. In *IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 94–103, Melbourne, Australia, May 2010.
45. S. P. Kavulya, K. Joshi, M. Hiltunen, S. Daniels, R. Gandhi, and P. Narasimhan. Practical experiences with chronics discovery in large telecommunications systems. In *ACM Workshop on Managing Systems via Log Analysis and Machine Learning Techniques (SLAML)*, Cascais, Portugal, October 2011.
46. T. Kelly. Detecting performance anomalies in global applications. In *USENIX WORLDS*, San Francisco, CA, December 2005.
47. G. Khanna, M. Y. Cheng, P. Varadharajan, S. Bagchi, M. P. Correia, and P. Verissimo. Automated rule-based diagnosis through a distributed monitor system. *IEEE Transactions on Dependable and Secure Computing*, 4(4):266–279, 2007.
48. G. Khanna, I. Laguna, F. A. Arshad, and S. Bagchi. Distributed diagnosis of failures in a three tier e-commerce system. In *IEEE Symposium on Reliable Distributed Systems*, pages 185–198, Beijing, China, October 2007.
49. E. Kiciman and A. Fox. Detecting application-level failures in component-based internet services. *IEEE Transactions on Neural Networks: Special Issue on Adaptive Learning Systems in Communication Networks*, 16(5):1027– 1041, September 2005.
50. R. R. Kompella, J. Yates, A. G. Greenberg, and A. C. Snoeren. IP fault localization via risk modeling. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 57–70, Boston, MA, May 2005.
51. J. Lala and L. Alger. Hardware and software fault tolerance: A unified architectural approach. In *IEEE FTCS International Symposium on Fault-Tolerant Computing*, pages 240–245, 1988.
52. P. E. Lanigan, S. Kavulya, T. E. Fuhrman, P. Narasimhan, and M. A. Salman. Diagnosis in automotive systems: A survey. Technical Report CMU-PDL-11-110, Carnegie Mellon University PDL, May 2011.
53. J. C. Laprie. Dependable computing: Concepts, limits, challenges. In *IEEE International Symposium on Fault-Tolerant Computing: Special Issue*, pages 42–54, 1995.
54. L. Lewis and G. Dreo. Extending trouble ticket systems to fault diagnostics. *IEEE Network*, 7(6):44–51, November 1993.
55. T. Y. Lin and D. P. Siewiorek. Error log analysis: Statistical modeling and heuristic trend analysis. *IEEE Transactions on Reliability*, 39:419–432, 1990.
56. M. Littman, N. Ravi, E. Fenson, and R. Howard. An instance-based state representation for network repair. In *19th National Conference on Artificial Intelligence (AAAI 2004)*, pages 287–292, July 2004.

57. G. Liu, A. Mok, and E. Yang. Composite events for network event correlation. In *International Symposium on Integrated Network Management*, pages 247 –260, Boston, MA, May 1999.
58. X. Liu, J. Heo, and L. Sha. Modeling 3-tiered web applications. In *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 307–310, Atlanta, GA, September 2005.
59. Z. Liu, B. Lee, S. Kandula, and R. Mahajan. NetClinic: Interactive visualization to enhance automated fault diagnosis in enterprise networks. In *IEEE Conference on Visual Analytics Science and Technology*, pages 131–138, Salt Lake City, UT, October 2010.
60. A. 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 *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 231–242, Barcelona, Spain, August 2009.
61. A. A. Mahimkar, H. H. Song, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and J. Emmons. Detecting the performance impact of upgrades in large operational networks. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 303–314, August 2010.
62. G. McCullough, N. McDowell, and G. Irwin. Fault diagnostics for internal combustion engines – current and future technologies. SAE Technical Paper Series 2007-01-1603, SAE International, April 2007.
63. P. McLachlan, T. Munzner, E. Koutsofios, and S. C. North. LiveRAC: Interactive visual exploration of system management time-series data. In *Conference on Human Factors in Computing Systems, CHI*, pages 1483–1492, Florence, Italy, April 2008.
64. A. V. Mirgorodskiy, N. Maruyama, and B. P. Miller. Problem diagnosis in large-scale computing environments. In *International Conference on High Performance Computing, Networking, Storage and Analysis*, page 88, Tampa, FL, November 2006.
65. G. Mongardi. Dependable computing for railway control systems. In *3rd IFIP International Working Conference on Dependable Computing for Critical Applications*, pages 255–277, 1993.
66. A. Oliner and J. Stearley. Bad words: Finding faults in Spirit's syslogs. In *8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008)*, pages 765–770, Lyon, France, May 2008.
67. A. J. Oliner, A. V. Kulkarni, and A. Aiken. Using correlated surprise to infer shared influence. In *IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 191–200, Chicago, IL, July 2010.
68. X. Pan, J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan. Blind Men and the Elephant: Piecing together Hadoop for diagnosis. In *International Symposium on Software Reliability Engineering (ISSRE)*, Mysuru, India, November 2009.
69. R. Patton. Fault detection and diagnosis in aerospace systems using analytical redundancy. *Computing Control Engineering Journal*, 2(3):127 –136, May 1991.
70. P. Peti, R. Obermaisser, and H. Kopetz. Out-of-Norm Assertions. In *11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS '05)*, pages 280–291, San Francisco, CA, March 2005.
71. M. Pizza, L. Strigini, A. Bondavalli, and F. Di Giandomenico. Optimal discrimination between transient and permanent faults. In *3rd IEEE International Symposium on High-Assurance Systems Engineering (HASE '98)*, pages 214–223, 1998.

72. F. P. Preparata, G. Metze, and R. T. Chien. On the connection assignment problem of diagnosable systems. *IEEE Transactions on Electronic Computing*, EC-16(6):848–854, December 1967.
73. P. Reynolds, C. E. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat. Pip: Detecting the unexpected in distributed systems. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 115–128, San Jose, CA, May 2006.
74. P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat. Wap5: black-box performance debugging for wide-area systems. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 347–356, New York, NY, USA, 2006. ACM Press.
75. I. Rish, M. Brodie, S. Ma, N. Odintsova, A. Beygelzimer, G. Grabarnik, and K. Hernandez. Adaptive diagnosis in distributed systems. *IEEE Transactions on Neural Networks*, 16(5):1088–1109, September 2005.
76. I. Rish, M. Brodie, N. Odintsova, S. Ma, and G. Grabarnik. Real-time problem determination in distributed systems using active probing. In *IEEE/IFIP Network Operations and Management Symposium*, pages 133–146, Seoul, South Korea, April 2004.
77. R. R. Sambasivan, A. X. Zheng, M. D. Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger. Diagnosing performance changes by comparing request flows. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 43–56, Boston, MA, March 2011.
78. B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. In *IEEE Conference on Dependable Systems and Networks*, pages 249–258, Philadelphia, PA, June 2006.
79. B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an MTTF of 1, 000, 000 hours mean to you? In *USENIX Conference on File and Storage Technologies*, pages 1–16, San Jose, CA, February 2007.
80. M. Serafini, A. Bondavalli, and N. Suri. Online diagnosis and recovery: On the choice and impact of tuning parameters. *IEEE Transactions on Dependable and Secure Computing*, 4(4):295–312, 2007.
81. K. Shen, C. Stewart, C. Li, and X. Li. Reference-driven performance anomaly identification. In *ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 85–96, Seattle, WA, June 2009.
82. D. P. Siewiorek and R. S. Swarz. *Reliable computer systems (3rd ed.): design and evaluation*. A. K. Peters, Ltd., 1998.
83. B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhagy. Dapper, a large-scale distributed systems tracing infrastructure. Technical Report dapper-2010-1, Google, April 2010.
84. L. Spainhower, J. Isenberg, R. Chillarege, and J. Berding. Design for fault-tolerance in system es/9000 model 900. In *in Proc. 22nd IEEE FTCS International Symposium on Fault-Tolerant Computing*, pages 38–47, 1992.
85. Splunk Inc. Splunk: The IT Search Company, 2005. http://www.splunk.com.
86. M. Steinder and A. S. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53(2):165–194, July 2004.
87. C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *European conference on Computer systems (EuroSys)*, pages 31–44, Lisbon, Portugal, March 2007.
88. A. W. Stroupe, S. Singh, R. Simmons, T. Smith, P. Tompkins, V. Verma, R. Vitti-Lyons, and M. Wagner. Technology for autonomous space systems. Technical Re-

port CMU-RI-TR-00-02, Carnegie Mellon University, Robotics Institute, September 2001.
89. J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan. SALSA: Analyzing Logs as State Machines. In *USENIX Workshop on Analysis of System Logs*, San Diego, CA, December 2008.
90. J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan. Mochi: Visual Log-Analysis Based Tools for Debugging Hadoop. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, San Diego, CA, June 2009.
91. B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi. An analytical model for multi-tier internet services and its applications. In *ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 291–302, Banff, Alberta, Canada, June 2005.
92. J. R. Wright and G. T. Vesonder. Expert systems in telecommunications. *Expert Systems with Applications*, 1(2):127 – 136, 1990.
93. W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting large-scale system problems by mining console logs. In *ACM Symposium on Operating Systems Principles*, pages 117–132, Big Sky, MT, October 2009.
94. S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High speed and robust event correlation. *Communications Magazine, IEEE*, 34(5):82–90, May 1996.
95. C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma. Automated known problem diagnosis with event traces. In *European conference on Computer systems (EuroSys)*, pages 375–388, April 2006.
96. S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox. Ensembles of models for automated diagnosis of system performance problems. In *IEEE Conference on Dependable Systems and Networks*, pages 644–653, Yokohoma, Japan, July 2005.