CAPTURING, INDEXING, AND RETRIEVING SYSTEM HISTORY

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Steve Yu Zhang
March 2007

Abstract

Complex networked systems are widely deployed today and support many popular services such as Google and Ebay.com. Due to their size and complexity, these systems tend to behave in ways that are difficult for operators to understand. In addition, frequent changes such as hardware and software upgrades mean that insights into system behavior could be invalidated at any time. When these complex systems exhibit problems, administrators must often analyze millions of metrics collected about system state, the vast majority of which are irrelevant for any particular problem. Furthermore, systematic methods of utilizing previous diagnostic efforts to aid problem resolution are lacking.

This dissertation describes our approach of automatically extracting indexable descriptions, or signatures, that distill the system information most associated with a problem and can be formally manipulated to facilitate automated clustering and similarity based search. We argue that our technique helps operators better manage problems both by improved leveraging of past diagnostic efforts, and by automated identification of relevant system information.

The first half of this thesis details how signatures can be used to aid system problem diagnosis and the methodology for evaluating their effectiveness. We also present a specific signature construction method based on statistical machine learning and show that signatures generated in this manner have significantly better clustering and retrieval properties compared to naive approaches. We validated our techniques on a testbed system with injected problems, as well as a production system serving real customers.

The latter half of this thesis focuses on a couple of challenges we faced. First, because system behavior is often highly dynamic, we introduce a technique for employing an ensemble of models to capture changes in behavior. Second, problem symptoms often depend on how normal system behavior is defined. We present a method of using multiple models of normality to make signatures robust to variances in normal system behavior.

We believe our signatures-based approach offers a promising framework for leveraging statistical and information retrieval techniques to address the challenges posed by the complexity of today's and tomorrow's systems.

Acknowledgements

First of all, I would like to convey my sincere gratitude to my Ph.D. adviser, Armando Fox. His invaluable advice and guidance have greatly helped me on my journey through graduate school. Armando's mentorship has been instrumental in my development as a computer scientist. I benefited enormously from our many discussions and I hope to be able to continue them in the future.

During the last three years of my research, I collaborated closely with Ira Cohen and Moises Goldszmidt at Hewlett-Packard Labs. Their work provided the basis for my thesis. This dissertation would truly not have been possible without their key contributions. My frequent interactions with Ira and Moises often inspired and enabled significant progress in my research. Since I entered graduate school with a background in systems, their statistical knowledge was critical to every major aspect of my thesis. As a member of my committee, Moises also offered many helpful suggestions for improving this dissertation. I appreciate all of the help and effort they have both put into working with me.

I would like to thank Christos Kozyrakis, who also served on my thesis committee. He not only provided valuable feedback for this thesis, but also devoted much time to help me improve my oral defense presentation. Thank you also to the other member of my orals committee, Philip Levis and Rob Tibshirani. Their insightful questions offered a helpful perspective regarding the potential and the limitations of my research.

I would also like to thank Kumar Goswami and Marilyn Lam for enabling my involvement with so many wonderful colleagues at Hewlett-Packard Labs. In particular, Julie Symons and Terence Kelly helped me immensely in collecting and preparing the datasets that were key to evaluating the approaches described in this thesis. Thank you to everyone else at HP Labs who have provided valuable feedback for several of my presentations and publications.

As part of the Recovery-Oriented Computing (ROC) project and then the Reliable Adaptive Distributed systems Laboratory (RAD Lab), I have had the pleasure of working with many graduate students and faculty at U.C. Berkeley. I appreciated all of the insightful presentations and research discussions at our various meeting and retreats. I would also like to thank several colleagues at Stanford who have collaborated with me, including George Candea, Pedram Keyani, and Emre Kıcıman.

Thank you to all of the staff member at the Stanford University Computer Science department, especially Sarah Lee and Kathi DiTommaso, who have helped me deal with an array of administrative tasks.

I would like to especially thank my parents, Rick and Ying, who have always put me and my education as their top priorities. They worked so hard and sacrificed so much in order to provide me with the opportunities they never had. I truly would not be the person I am today without their love and support.

Most of all, I want to express my deep love and appreciation for Shuyi, my wife and my best friend. She has always loved, supported, and motivated me in countless ways. She inspires me to be a better person in every aspect. I am especially grateful for the spirit, joy, and happiness that she continually brings into my life.

During my career as a Ph.D. student, I have received generous financial support from the National Science Foundation as well as Hewlett-Packard Labs.

Contents

List of Tables

List of Figures

Chapter 1

Overview and motivation

The continuous growth in computer processing power and network capabilities over the last 50 years have enabled the construction of extremely complex networked systems, which are deployed widely today. Trillions of dollars of stocks and bonds are traded everyday through electronic means. Billions of emails, which has become closely intertwined in both our business and social lives, are exchanged every hour. Millions of inquiries are answered by Internet search engines each minute. It is clear that the very fabric of our society has come to depend on the correct functioning of these types of systems. However, proper management of these very complex systems presents a daunting challenge. In the relatively short history of computing, designing for reliability has usually taken a backseat to maximizing raw performance. The unfortunate consequence is that not only are systems today prone to many types of failures, but also that when problems do occur, they are often difficult to diagnose and resolve.

1.1 Problem: diagnosing failures in complex systems

1.1.1 System (un)reliability

Although computers today are hundreds of times more powerful compared to those of 15 years ago, their reliability has not improved nearly that quickly. While electronic hardware today is more reliable, the ever increasing complexity of computer systems has resulted in more points of failure for each system. In addition, while programming languages and design techniques have improved in recent history, there are still many bugs that make it into a production system. Popular online services such as Google and Amazon.com utilize data centers with ten of thousands of components, ranging from servers to network attached storage to routers and switches. Misbehavior of even a small subset of these components can have significant impact on the performance and availability of the entire service. Configuring and upgrading the software for these systems are also extremely complex tasks that sometimes cause unanticipated downtime or service degradation.

According to Keynote, an Internet service monitor, the top 40 business web sites had a typical availability of 98% during 2006. This translates to roughly 7 days of downtime each per year. Contrast this to telephone networks, where the gold standard is "five nines" availability (99.999%), or less than 5 minutes per year [32]. Problems[1] are not restricted to large Internet services. During two days in mid-January 2007, the Chicago Board of Trade suffered 3 separate outages of their electronic trading system [45]. The economic cost of a slowdown or outage extends beyond just the revenue lost during the problem. Over the last year, service unavailability at high profile services such as Ebay.com, Youtube.com, Amazon.com, and many more, were widely reported by the mainstream press. Although most of these failures only resulted in a few hours of downtime, the negative publicity from such an incident can affect

[1]We use the term problem to refer broadly to any undesirable behavior of a system, regardless of root cause. This includes fail-stop scenarios that render a service completely unavailable, situations where the service is merely unacceptably slow, and cases when the system responds incorrectly to given input. See section 2.2.2 for a more detailed discussion of undesirable behavior.

business well after the original issue is fully addressed.

1.1.2 Complex systems difficult to understand

Although diagnosing problems in simple single node systems is often far from trivial, the sheer size and complexity of large scale services, many designed to serve up to millions of concurrent users, presents several additional significant challenges.

- Large complex systems exhibit emergent behavior that is not easily predictable from the behavior of individual components [37]. The more complex the system, the more likely that there is emergent behavior.
- It is difficult for any single person to be familiar with all of the components of a large system and how they interact with one another. Often, these services are supported by very large teams of engineers, where each individual understands only a small aspect of the system in detail. Further complicating matters, separate divisions of a company or even separate companies can be responsible for different parts of a system. Responsibility for resolving issues is frequently passed back and forth from one administrative unit to another, hampering diagnostic and repair efforts.
- Many large systems and the environments in which they operate are continually evolving. Even if it were possible to analyze and understand all of the intricacies in a complex system, much of the analysis would often be invalidated by software or hardware upgrades to even a single component, as well as by significant changes in the nature of the workload applied to the system.

These challenges apply not only to problem resolution, but also to the initial deployment and ongoing maintenance of large complex software systems.

1.1.3 Current diagnosis techniques are primitive

The first step towards resolving failures is to detect them. Failure detection is difficult and much work that has been done recently in this area [11, 27]. However, the focus of this thesis is on the second step, diagnosing the problem once it is discovered. While simple actions, such as rebooting a system in part or whole [9], may solve a variety of problems without requiring full diagnosis, many problems require detailed analysis to determine permanent solutions that should be applied. This includes conditions related to misallocation or shortage of resources that leads to persistent degradations in performance and other anomalies that can be addressed only by nontrivial configuration changes.

Failure diagnosis today tends to rely on knowledgable system administrators. They must be familiar with all of the important components and have a solid higher level understanding of the behavior of the system as whole. Past experience in diagnosing and fixing problems in their system or a similar system usually helps them to resolve common problems quickly. Although this approach might be adequate for an experienced operator managing a system with only a few nodes, it is a poor solution in general for several reasons.

- Capable and experienced system administrators are difficult to find and expensive to keep. Partly (if not mostly) due to the lack of advanced tools for failure analysis and recovery, good system administrators are in high demand.
- Diagnosing a problem that has never been seen before still presents a challenge, even to the most astute operators.
- Most importantly, this approach does not scale well to extremely complex systems. A system is more than just the sum of its parts. Having a group of administrators, each responsible for a portion of a system, is simply not effective for diagnosing systemic issues.

Furthermore, there are currently only informal techniques for leveraging past diagnostic efforts. Using the text of error messages to search the World Wide Web is one common approach. However, those methods are inadequate since many problems do not exhibit simple error messages. Therefore, we believe that a systematic method for identifying and retrieving similar problem instances from the past would be invaluable. If the problem had been previously resolved, we could simply revisit the diagnosis and perhaps reapply the repair actions. Even if the past problem remained unresolved, we could gather statistics regarding the frequency of recurrence of that problem, accumulating vital information for prioritizing or escalating diagnosis and repair efforts.

1.1.4 Too much information

While it may appear that the predominant problem for novice troubleshooters is the lack of knowledge about system behavior, in many cases, what is truly hampering their efforts is actually the overabundance of information, most of which being irrelevant. Besides collecting high level data about the behavior of the overall service, e.g. throughput and latency, many metrics[2] from multiple layers of abstraction are usually monitored for each component of a system. These typically include:

- Hardware level metrics (e.g. power consumption)
- Networking metrics (e.g. packet rates)
- Operating system level metrics (e.g. CPU utilization, memory usage)
- Application specific metrics (e.g. event counts)

In addition, application and operating system messages are usually dumped into log files [3]. Section 2.2.1 examines in more detail all of the types of information that is typically collected about systems.

The intuition that good administrators develop over time allows them to quickly reduce the universe of possible diagnosis to a few prime suspects, given a set of symptoms about a problem. Information about the system that cannot be used to confirm or contradict those suspicions can simply be ignored unless all of those suspicions prove false. However, for larger and more complex systems, experience alone becomes inadequate for filtering the deluge of mostly irrelevant data into a manageable stream of important information.

[2]Usually up to a few hundred per system node.

Frequently, even more information could be monitored or logged. The more data that is collected, the greater the chance that details important for accurate diagnosis are included. Unfortunately, it is already impractical for even a large group of operators to wade through the enormous amounts gathered data, often exceeding hundreds of Gigabytes per day. Filtering for potentially relevant information is like sifting for needles in a haystack. The ease with which computers can analyze large collections of information, such as in data mining, suggests that designing automated techniques to aid system problem diagnosis may be a fruitful approach. Furthermore, given the dynamic and ever evolving nature of systems today, techniques which rely on detailed specifications about the structure of a system would be impractical. Such specifications would be expensive to generate and require constant updating to reflect system changes. Any approaches we develop should be useful without *a priori* system specific knowledge.

1.2 Approach: constructing signatures of system state

Our approach involves automatically extracting *indexable* descriptions, or *signatures*, that both distill the system information most associated with a problem and can be formally manipulated to facilitate automated clustering and similarity based search. We argue that our technique helps operators better manage problems both by improved leveraging of past diagnostic efforts, and by automated identification of relevant system information.

Diagnosing problems in the large software systems ubiquitous today requires techniques that can leverage large amounts of observed data and help operators understand the many complex behaviors that their systems may exhibit. This dissertation describes our approach of utilizing statistical machine learning methods to extract indexable signatures of system states. We will also explore the various ways in which operators can harness the power of signatures to aid diagnosis and resolution of system failures.

A signature is a indexable representation of system state that captures information relevant to high level system behavior, where similarity between states can be inferred by the distance between signatures. The more accurate the inference – we refer to this as the accuracy of the signatures – the more useful the signatures. Perfect accuracy is not required for signature usability. Nor is there a well defined boundary where signatures go from useful to not useful. While the marginal utility of increasing signature accuracy is an interesting topic, it will not be addressed in this thesis and we simply assume that higher signature accuracy is always better. Using accurate signatures offers the following benefits:

- A systematic way of comparing system problems by comparing their signatures. Past diagnostic efforts can be applied whenever signatures indicate that a problem is similar to a previous one. Diagnostic efforts are often quite expensive, meaning that the cost savings from being able to leverage past work may be enormous.

- The ability to gather statistics on the frequency of recurrence of each problem. This gives operators the ability a sound method for prioritizing or delegating problem diagnosis efforts.

- Although it is not a requirement that signatures must have a smaller memory footprint than the original system data, practically speaking, this should be case for any reasonable method of generating the signatures. If that is the case, then signatures also represent an efficient method of archiving system history. System history often has to be stored for long periods of time not only to aid system management, but also for auditing or compliance needs. Signatures not only provide for the ability to index system history, they may also be stored *in lieu of* the real system data, often resulting in storage cost savings.

Note that these benefits apply to *any* signatures based approach, regardless of how the signatures themselves are actually extracted. Signatures could be created manually, i.e. by operators sifting through system data. We will present not only ways of utilizing signatures (regardless of how they are created) for system problem diagnosis, but also an automated statistical method for extracting signatures that results in significantly more accurate signatures when compared to naive approaches.

We use statistical machine learning techniques to extract relevant patterns and correlations from information collected about a system. These patterns are then encoded to become the signature of the state that the system was in when those observations were recorded. In the past, these methods were often considered too computationally expensive for use in a soft-real time manner. However, significant improvements in both processing power and the statistical methods themselves have enabled analysis of system data in soft real time.

There are three main advantages of our statistical approach.

- No assumptions are made about system design, structure, or behavior. Because *a priori* knowledge is unnecessary, this technique can be applied to almost any kind of system. Also, adaptation to changes in system internals or workload is natural and no operator invention is needed.

- Since relevant patterns and correlations are encoded in signatures themselves, operators can directly use this information to help with diagnostic efforts.

- The approach is fast enough to allow for soft-real time analysis, where signatures of system state can be generated seconds after monitored system data is recorded.

For any method based on statistical extraction of patterns and correlations to reasonably work, relevant patterns and correlations must actually exist and be captured by the data collected from a system. Fortunately, the primary reason more information is not usually captured about a system is the lack of ability to analyze the additional data. Our automated statistical techniques should address that concern and encourage administrators to expand monitoring capabilities, which will in turn improve the potential usefulness of our approach.

1.3 Contributions and thesis map

This thesis presents a statistical approach to analyzing failures in complex software systems. In particular, we are advocating using statistical and machine learning techniques to automatically generate signatures of system state. The goal of signatures is to aid operators (especially less experienced ones) in diagnosing and resolving system problems.

The first half of this thesis explains the background and techniques for constructing and using signatures and demonstrates their effectiveness. The latter half explores a couple of fundamental challenges we encountered. These challenges apply not only to the approach we advocate, but also to many other statistics based approaches for this domain.

This thesis makes four main contributions:

- An method for automatically generating signatures of system state that capture the essential state of an enterprise system and are effective for clustering and similarity based retrieval using known techniques from pattern recognition and information retrieval [18]. We show that the construction of an effective signature is nontrivial – the naive approach yields poor clustering and retrieval behaviors, but good results are obtained with an approach based on the use of statistical methods to capture relationships between low-level system metrics and high-level behaviors. We also show that this approach is computationally efficient enough to be applied in soft-real time in large-scale systems.

- Demonstration of using signatures to cluster and identify performance problems, and to compute statistics about the frequency of their occurrence. This in turn lets an operator distinguish a recurrent condition from a transient or first-time condition, and even annotate the corresponding signatures(s) with a repair procedure or other explanation for future reference when the same problem recurs.

- A signature generation method that allow for adaptation to changing system and workload behavior by employing an *ensemble* of statistical models.

- Because system problem symptoms depend on the desirable behaviors that a system exhibits, we developed a technique for evaluating and improving the robustness of signatures with respect to variances in desirable behavior.

All of our techniques were verified on data from both an experimental testbed as well as an enterprise system.

Chapter 2 presents background information. We first describe the common class of enterprise systems that we focused on for the evaluation of our approach. We then state some assumptions about these systems and their behaviors. Finally, we will introduce statistical machine learning concepts that will be used in explaining our approach.

Chapter 3 describes how signatures can be used and evaluated while Chapter 4 describes in detail our specific method for generating signatures. Chapter 5 presents our evaluation datasets and the results of the evaluation.

Chapters 6 and 7 explore significant challenges that we faced. First, we describe our method for building models based on ever changing system behaviors. We then discuss the how lack of a single model of desirable system behavior complicates signature construction and usage, and a method for mitigating those effects.

Chapter 8 examines the usage of a signature based diagnosis tool from an system operator's point of view and discusses limitations of our technique and statistical techniques in general.

Chapter 9 explores related work in system problem detection and diagnosis. We also describe signature based approaches for tasks in other domains.

Finally, Chapters 10 and 11 describe future work and conclude.

Chapter 2

Background

We define a signature to be an *indexable* description of system state. Ideally, similarity between signatures directly implies similarity between the root causes of the corresponding system states. The accuracy of this implication determines the quality of the signatures. Good signatures offer operators a systematic method of comparing system problems. This allows them to easily leverage previous diagnostic efforts when new failures emerge, among many other benefits that signatures can provide.

This dissertation advocates a method of automatically generating signatures using statistical machine learning (SML) techniques. This chapter will first describe a popular class of systems, the three tiered internet services. We will then explain our assumptions about systems and system behavior that enable our approach. Finally, some important SML concepts that form a basis for our approach are explored.

2.1 Three tiered Internet services

During their infancy in the early 1990s, online services tended to be single machine web servers delivering static content. Today, popular websites, such as Ebay, Amazon.com, and Google, utilize interconnected networks of tens or hundreds of thousands of nodes, geographically distributed around the globe [6], usually configured as a three-tiered system.

The three tiers consist of the following:

- The top layer of this tiered structure is the presentation layer. This tier consists of web servers, which are responsible for serving static content, forwarding user requests for dynamic content to lower layers, and performing the processing necessary to format the lower layer's responses to those requests. Servers in this tier usually run web server software such as Apache HTTP Server, or Microsoft's Internet Information Services(IIS).

- Underneath the presentation layer is the application logic tier. Servers that comprise this layer are responsible for the business logic that usually represents the core functionalities of an Internet service. For example, it would be the role of Ebay's application logic tier to verify the validity of users' bid requests and transform them into a proper sequence of actions such that the new bid is consistently reflected in the database[1].

  Application logic is usually designed to operate on top of middleware, typically a J2EE application server such as Weblogic (BEA), Jboss (Red Hat), WebSphere (IBM), or Microsoft's .NET platform. Middleware runs on top of servers' native operating system and provides generic functionality such as security and data integrity. This allows companies to focus only on their business specific features when developing application logic tier software.

- The bottom layer is referred to as the storage or database tier. These servers are responsible for managing the persistent data needs of the service. Oracle and MySQL are popular choices for database software.

These types of systems are mainly comprised of clusters of servers, with each cluster serving one of three roles. Each cluster tends to consist of nearly identical hardware running similarly configured software and a load balancer to ensure every server is equally utilized. Large Internet services should vigorously stress any automated diagnostic approach. The complex systems that support these services are good evaluations cases for the following reasons:

[1]To improve performance, sanity checking and some business logic may be done at the presentation layer. Likewise, persistent data from the storage layer may be cached at the higher layers to mitigate the high latencies usually associated with database accesses.

- Hardware and software changes are inevitable for almost all large systems. However, they occur especially often for Internet systems. Many large services roll out minor changes almost every day and major updates every month. This is due to their extremely fast pace of growth as well as the constant need to offer new features and fix bugs.
- These systems often exhibit recurring problems due to configuration issues. This is because they are often comprised of hardware and software from many different vendor that may not always be very compatible with one another.
- The commodity hardware on which most of these systems rely tend to fail more often than top of the line hardware. Because very large services utilize tens or hundreds of thousands of individual machines, the cost of using only the most reliable hardware is usually prohibitive.

Another main reason we focus on these systems is that they are generally consistent with the assumptions that we describe in the next section, which are prerequisite for useful signatures.

2.2 Assumptions about systems and system behaviors

Our approach relies on finding statistical patterns within system data. In particular, we assume that there is a binary high level indictor of system state, the *service level objective* or SLO. At any time, a system is either in *compliance* of or in *violation* of its SLO. In addition, we assume that many low level metrics (e.g. CPU utilization) are collected and that these metrics are relevant to the SLO state. We aim for signatures to represent the relationships between a system's low level metrics and high level behavior. Consequently, using SML techniques requires that these relationships be able to be captured by statistical models. The rest of this section explores these assumptions in more detail.

Most Internet services are governed by simple SLOs based on average transactional response times. In addition, large Internet systems, such as those employed at Amazon.com [3], tend to be completely instrumented and monitored at a very fine-grained level. Often, additional instrumentation is possible but not currently employed due to the lack of ability of analyzing the data. Finally, many common problems that these systems exhibit can be characterized by relatively simple patterns of behaviors in the low level metrics.

2.2.1 Measurable system properties

There are two types of measurements that are typically collected about a system. Properties about behavior of a system as a whole are one type. We will refer to properties of this type as *high level metrics*. They usually measure the correctness, efficiency, or availability of the service that the system is providing. Average response time and throughput are the two predominant measurements of overall system performance. However, high level behavior properties also encompass metrics such as failure rate and click-thru rate, among many others. The second type of measurements, which we will refer to as *low level metrics*, usually reflects the state or activities of individual components of a system. This might include the CPU utilizations of application servers, the packet counts of routers, or the free disk space of storage units. Table 2.1 provides more examples of both measurement types. Note that while it tends to be clear with higher level measures whether larger or smaller values are generally better, this is often not the case with lower level metrics. This insight is one of the reasons that naive approaches to signature construction are ineffective.

While the high level metrics are typically captured in a variety of ways depending on the exact metric and specific system, low level metrics are usually captured by software such as HP OpenView or Ganglia [22, 35]. Metrics are typically recorded every 15 seconds to 5 minutes. We refer to this period as an *epoch* and epoch length is usually configurable. Since the collection and reporting of measurements usually has a negative impact on performance, it is undesirable to measure too frequently. However, this must be balanced by the consideration that too coarse of measurements

Metric Name	**Metric Description**
	High Level Metrics
Average response time	Average latency of user requests
Click-thru rate	Rate of site visitors clicking on advertisements Or rate of search engine users clicking on search results
Average session length	Average length of time a user spends at a site
User satisfaction rate	Results of a live online survey about a service
	Low Level Metrics
tt_count	Total transaction count
gbl_cpu_total_time	Total CPU time spent on all processes
gbl_active_cpu	The number of CPUs online in a server
gbl_run_queue	The average number of 'runnable' processes
gbl_syscall_rate	The number of system calls per second
gbl_mem_pageout	The total number of page outs to the disk
gbl_net_in_packet_rate	Number of successful packets per second received
fs_space_util	Percentage of file system space in use
bydsk_phys_read	Number of physical reads for disk device
bycpu_interrupt_rate	Average number of IO interrupts per second

Table 2.1: **Examples of high and low level metrics.** The names of the low level metrics are the actual names of the metrics used by a popular system monitoring software.

have little value when attempting to find useful patterns and correlations within them.

2.2.2 Service level objectives

Performance of complex networked systems are often bound by a service level objective or SLO. This is also well known as a service level agreement, which also usually refers to the contractual agreement that specifies the service level objectives. Objectives are typically based on one or a combination of a system's high level metrics. For example, a SLO might stipulate that the average response time of a search engine service be below 4 seconds while retaining a click-thru rate above 10%. They might represent goals internal to an organization, e.g. a promise from the IT department to corporate headquarters. However, they are also often included in legally binding contracts in which violations of the service level objective can have severe financial repercussions. This is commonly implemented when a company outsources a consumer-facing service to a third-party vendor, which must be able to ensure a positive experience for that company's users. It may also represent the formalization of a guarantee that an organization makes directly to its customers.

Although the exact SLO that is specified for a system carries substantial design and administrative implications, SLOs could be determined rather arbitrarily, at least from a system operator's point of view. When violations of a SLO occur, it does not necessarily imply that there is bug, misconfiguration, or component failure. It may indicate that the system was subjected to an atypically high workload. It is also possible that the SLO criterion is unrealistic or inaccurately measured. We will discuss the implications of using an arbitrary SLO in section 8.2.1. However, no matter how the service level objective is determined, the onus is still usually on system operators to ensure that SLO violations occur as infrequently as possible.

We utilize the SLO as a systematic way of separating the overall state of a system into two classes, *SLO compliance* and *SLO violation*. We will also refer to *SLO compliance* as *SLO non-violation*. Although the behavior of a system is rarely binary, exactly how the system is behaving is largely unimportant if it is always in a state of SLO compliance[2]. Conversely, excuses about complex system behaviors are rarely acceptable when there are frequent SLO violations. Our approach will take advantage of this convenient binary delineation of desirable versus undesirable system behavior.

2.2.3 Relationships between low level metrics and SLO

Since signatures try to capture relationships between low level metrics and a system's high level SLO state, those low level metrics must be relevant to the overall system behavior reflected by the SLO state. For example, if problems occur that are related to usage of a particular network interface and no metrics from that interface are collected, then signatures will likely be of little use. Furthermore, we assume that the relevant patterns and correlations between system metrics and high level SLO state can be accurately captured by a statistical model. Note that we are not attempting to explicitly model exact system behavior. For example, we do not try to predict the average response time of a service based on low level metrics alone. All we need to do for this approach to be successful is to capture patterns in low level metrics that separate clearly defined desirable system states (SLO violations) from undesirable system states (SLO compliances).

2.3 Statistical machine learning overview

Statistical machine learning (SML) algorithms have been applied in a wide range of domains for many decades. Some examples of these applications include natural language processing, medical diagnosis, bioinformatics, credit card fraud detection, stock market analysis, DNA sequence classification, and robot locomotion [46]. Only recently has it been applied to the area of systems management. In this section, we will use a common and successful application of SML, for handwriting recognition, to help introduce some concepts and terms that will be used throughout this thesis.

[2]A key exception being when it is expected that the workload to a system will change drastically in the future and the operators need to anticipate the reaction of the system to those changes.

2.3.1 Supervised learning and classification

When hand addressed parcels are given to a local post office, machine sorters scan and interpret the address and direct the mail to the proper bin to continue its journey to its destination. Given the wide range of handwriting styles and the haste which with mailing addresses are often written, it is an impressive feat that machines are able to recognize the vast majority of these addresses. Applying statistical learning to this problem is straightforward. First, a learning algorithm is provided a *training set* of data that consists of handwriting samples and the correct interpretations of those samples. The "correct" interpretation here is usually assumed to be the interpretation of a human handwriting expert and is generally referred to as the *ground truth*. This process uses the most mature type of SML algorithms, those for *supervised learning*. In *unsupervised learning*, no training set with the ground truth is provided.

More specifically, this problem is a formulation of a very common supervised learning task, that of *classification*. The algorithms for classification are usually referred to as *classifiers*. For classification tasks, classifiers try to learn the approximate behavior of a function that maps an input vector (also known as a *feature vector*) into one of several classes. Technically, the number of classes is not limited as long as it is finite. However, for most classification tasks, the number classes is less than a hundred. The training set consists of samples of input vectors and their corresponding correct classifications. For handwriting recognition, the input feature vector represents the letter to be interpreted and the classification output should be one of the 36 alphanumeric possibilities for that letter.

After processing the training data, a classifier can be used to predict the proper classification of input vectors (handwritten letter samples). Prediction is used throughout this dissertation in the sense of interpolating or extrapolating some property that we are not given. It is not necessarily referencing an event that has yet to occur. That we refer to as *forecasting*. The *classification accuracy* of the algorithm is evaluated by how well its output matches up with the ground truth. The data used for evaluation purposes is usually referred to as the *test set*. In general, if there are no significant differences, e.g. only samples from left handed writers in the training set but only samples from right handed writers in the test set, between the training set and test set, the more training data an algorithm is given, the better the algorithm will perform when evaluated using the test set. However, the relationship is not usually linear. Rather, when given very little training data, accuracy usually improves drastically if given additional training data. However, the marginal improvement to accuracy generally decreases as more and more training data is processed. Also, not only does using more training data have a greater computational cost, the availability of data with the corresponding ground truth is often limited. Therefore, these tradeoffs must be balanced when determining the optimal size for a training set.

The ultimate goal of a training a classifier is usually to use it to classify samples whose ground truth is unknown[3]. However, the precise accuracy with which an trained classifier correctly classifies these samples is unknown. Evaluation requires ground truth and having the proper classifications already obviates the need for using a SML algorithm. Fortunately, much of statistics is based on the premise that one can extract meaningful predications about a large population based on a much smaller sample. It is assumed that when an learning algorithm is able to accurately evaluate samples in test set, useful patterns or correlations must have been modeled by the algorithm. However, it is possible that the algorithm actually learned nothing and merely "lucked into" the correct classifications. While this is always a statistical possibility, the larger the test set and the more representative it is of the entire population, the less likely that good accuracy can be contributed to mere "luck".

2.3.2 Bayesian network classifiers

Bayesian network classifiers are widely employed for classification tasks today. One of the key properties of this class of models is interpretability. This means that not only can we use this type of model to determine the proper classification of samples, we can also interrogate the model for meaningful patterns and correlations that the model has learned. Other popular methods such as neural networks and kernel functions are not easily interpreted and can be used for their predictive value only. In relation to the handwriting recognition example, if we employed a Bayesian network classifier

[3]The main notable exception being in SML research, where the goal is merely to compare and evaluate new learning algorithms

for that task, we could interrogate the classifier to determine exactly what values of individual features make a sample likely to be any particular letter. Note however, that even if a classifier is used in this manner, the prediction accuracy is still important as an indictor of the strength and relevance of the patterns and correlations captured by the model.

Bayesian networks are computationally efficient representational data structures for probability distributions [41]. Since complex system may not always be completely characterized by statistical induction alone, Bayesian networks have the added advantage of being able to incorporate human expert knowledge. The use of Bayesian networks as classifiers has been studied extensively [21]. The simplest and most common Bayesian network approach is the naive Bayes classifier, which assumes complete independence among all of the features of the input vector. Although this assumption rarely holds in reality, naive Bayes classifiers have proven to be effective and extremely efficient at a variety of tasks, for example as a Spam filter [47]. At the other end of the complexity spectrum, full Bayesian networks allow for all possible relationships between features. However, this flexibility makes inducing these classifiers difficult and computationally expensive. The specific Bayesian network approach advocated in [15] is a compromise known as Tree Augmented Naive Bayes (TAN) models. TAN models allow each input feature to have one or zero child to parent relationships to other features. The benefits of TAN and its performance in pattern classification is studied and reported in [21].

2.3.3 Clustering

An unsupervised learning method, data clustering, will also be referenced in this thesis. We use this technique later not for constructing signatures, but rather way of leveraging signatures for diagnostic efforts. Clustering refers to the process of partitioning a dataset into subsets or clusters, so that the data in each cluster ideally share some common properties. In particular, clustering algorithms aim to minimize the average distance from each data point to its cluster center, for some defined distance measure. These algorithms are typically iterative such that some initial partitioning of the data is assumed and this partitioning is evaluated and improved repeatedly until a threshold is reached. The most common methods in this domain are hierarchical clustering and k-means clustering. Hierarchical algorithms either successively builds up or breaks up clusters in order to minimize the average distance to cluster centers. On the other hand, the k-means algorithm always forms k clusters and each iterative step refines the location of those k cluster centers to achieve the same goal of minimizing distance to those centers.

Chapter 3

Signature usage and evaluation methodology

Enterprise systems today are complex. When they exhibit undesirable behavior, whether it be poor performance, partial failures, or incorrect behavior, the cause of the problem tends to be difficult to diagnose, even for very experienced operators. Typically, ad-hoc trial and error approaches are used as the main root cause determination and resolution strategy. Such approaches are often time consuming and especially challenging for novice troubleshooters. What usually makes diagnosis difficult is not the lack of information about a system, but the overwhelming amount of data describing a system's behavior. Only a small subset of data is relevant to diagnosing any given problem, but there are no advanced tools to aid operators in identifying this subset.

In addition, today there is no systematic method for leveraging past diagnostic efforts when new problems arise. Hence expensive diagnostic efforts are often needlessly repeated. Therefore, we advocate the approach of automatically creating indexable signatures of system state that facilities similarity based search and retrieval. Ideally, signatures should not only offer a systematic way of inferring the similarity between two system problem states, but also help operators identify the subset of data that is likely relevant to problems.

While the next chapter will detail a specific method for generating signatures, the rest of this chapter will focus on how to utilize and evaluate signatures in general, regardless of how they are constructed. The work described the next three chapters was previously published in [15, 53, 16] and was jointly done with Ira Cohen, Moises Goldszmidt, Julie Symons, and Terence Kelly of Hewlett-Packard Labs.

3.1 Overview

As a system operates, data such as those described in 2.2.1 are collected periodically. The first step towards using signatures is to construct them using the captured system information (detailed techniques for doing this are presented in the next chapter). As signatures are built, they should be stored in a database.

System operators may then use a database of signatures in two main ways: clustering and retrieval. Clustering attempts to find a natural groupings of signatures that characterize different system problems, which offers operators a way of prioritizing efforts when many problems have yet to be diagnosed. Retrieval aims to find the signatures that are the closest to some search signature and thus most likely to represent "similar" problems. Retrieval is most helpful when many past problem have been diagnosed and an operator seeks to determine if a new problem can be identified as one of those previous issues. Both usage methods depend on defining the distance between signatures. We define signatures to be vectors of natural numbers. Therefore, any vector distance formula can be applied.

Since there is no ground truth for what a system signature should be, we evaluate signature accuracy based on if similar signatures represent similar system problems. We represent this accuracy as a *precision-recall* graph, which is commonly used in the information retrieval domain. However, in order for this evaluation to be possible, signatures must be *labeled* with their corresponding problem so that we can determine if two signatures are supposed to represent the same issue or not. However, labels are determined by human operators and are therefore usually inexact, incomplete, and expensive to obtain. In the absence of labels, partial evaluation is still possible by examining signature clustering to determine if the signatures at least capture information relevant for separating SLO compliance and SLO violation.

3.2 Calculating distance between signatures

The main usage methods for signatures, clustering and retrieval, are described in the next two sections. However, both techniques depend on formalizing the notion of similarity between signatures. Since signatures are just vectors of numbers, any common vector *distance metric* could be used. We explored three such metrics. Let $\vec{S^1} = [a^1_1, \ldots, a^1_n]$ and $\vec{S^2} = [a^2_1, \ldots, a^2_n]$.

- **Cityblock distance**: Also known as the L_1 norm, this distance between $\vec{S^1}$ and $\vec{S^2}$ would be $\sum_{i=1}^{n} |a^1_i - a^2_i|$.
- **Square Euclidean distance**: Also known as the L_2 norm, this distance between $\vec{S^1}$ and $\vec{S^2}$ would be $\sqrt{\sum_{i=1}^{n}(a^1_i - a^2_i)^2}$.
- **Cosine distance**: This is one minus the cosine of the angle between two vector and between $\vec{S^1}$ and $\vec{S^2}$ would be $1 - \frac{\sum_{i=1}^{n} a^1_i a^2_i}{\sqrt{\sum_{i=1}^{n} a^1_i a^1_i}\sqrt{\sum_{i=1}^{n} a^2_i a^2_i}}$

This thesis includes only results based on the cityblock distance because it is the most efficient to compute. Although the results when using the square Euclidean and cosine distances were quantitatively different, they were qualitatively similar and offered no new insight as far as making decisions about signature composition or other parameters of the process.

3.3 Retrieving signatures

Using information retrieval techniques, we can search a database of signatures for the previous instances that are closest (in terms of the distance measure described in the previous section) to a specific signature $\vec{S}$. This capability enables operators to leverage past diagnosis and repairs and generally all information about previous instances displaying similar characteristics (as captured in the signature vector).

For evaluation, we will follow the standard measures from the machine learning and information retrieval community [51]. In the information retrieval domain, a search retrieves some number of entries from a database of documents. Each search result is considered either irrelevant or relevant given the search terms[1]. The more relevant entries returned (without returning extra irrelevant items) the better. This implies the need for ground truth about the relevance of each item in a database with respect every possible search term. In our case, this relevance is derived from the *label* (or *annotation*) applied to each signature. Presumably, the labels are associated with a root cause or resolution method. If we search for the closest signatures to a *test signature*, a search result is considered relevant if it has the same label as the test signature, and irrelevant otherwise.

Since the ratio of relevant to irrelevant search results varies depends on the size of the result set, and it is important to vary search parameters to fully gauge the success of the method, we present retrieval results using *Precision-Recall* (PR) curves. *Precision* is defined to be the percentage of search results that are relevant. *Recall* is defined to the number of relevant search results divided by the total number of relevant signatures in the database. In general, recall increases and precision decreases as the search result size increases. Ideally, precision would always equal one. If retrieval behavior needed to be described using a single number, the area under the PR curve should be used. The larger that number the better, with the best curve having an area of one.

We construct PR curves separately for each distinct signature label in the database for two reason. First, retrieval behavior can vary significantly depending on the type of problem used as the test signature. More importantly, this allows evaluation using data sets where not all signatures are labeled. In order to derive the PR curve for problem label L, we only need to know if each signature has label L or not. If it was not L, we do not need the proper label. It is common for operators to know all occurrences of a particular problem but be unsure about the cause of other problem instances. The exact procedure for generating a precision-recall graph is described in Algorithm 1.

[1]Relevance need not be binary.

Algorithm 1 Generating a Precision-Recall curve for signature label L

```
Let $t$ denote the total number of signatures in the database
Let $a$ denote the number of signatures in the database with label $L$
Let $n$ denote the size of the result set
Initialize $p_n = 0$ and $r_n = 0$ for $(n = 1, \dots, t-1)$
for test signature = each signature with label $L$ do
  for $n = 1$ to $(t-1)$ do
    Find the $n$ signatures in the database closest to the test signature (sec 3.2)
    Let $m$ of those $n$ signatures also have label $L$
    $p_n = p_n + \frac{m}{n}$
    $r_n = r_n + \frac{m}{a-1}$
  end for
end for
for $n = 1$ to $(t-1)$ do
  $p_n = \frac{p_n}{a}$
  $r_n = \frac{r_n}{a}$
  add $(r_n, p_n)$ to curve {Recall is always on the x-axis and Precision on the y-axis}
end for
```

3.4 Clustering signatures

The objective when applying clustering to a database of signatures is to find the natural groupings or clusters of these signatures that characterize different system problems (and optionally normal operation regimes). The output of clustering is a set of clusters, plus a characterization of each cluster center. By inspecting the actual elements of the signature database in each cluster, we can identify different regions of normality as well as recurrent problems. In addition, the centroid of a cluster of problem behaviors can be used as the *syndrome* for the problem, since it highlights the metrics that are in a sense characteristic of a set of manifestations of the same problem.

3.4.1 Clustering algorithms

Clustering requires not only a distance metric(see Section 3.2, we again use only show results using cityblock distances), but it also requires specification of a clustering algorithm, which attempts to the minimize distortion (sum of distances of each signature with respect to its nearest cluster center). We use the standard k-means iterative algorithm [18]. This algorithm finds k cluster centers that minimize the distortion defined above. Other methods such as hierarchical clustering were also explored with no qualitative difference in results.

3.4.2 Clustering entropy

Ideally, we would like each cluster to contain signatures belonging to a single class of problems (i.e. SLO violations with the same root cause), or else signatures belonging only to periods of SLO compliance (when clustering signatures of violations and compliance periods together). We introduce a score determining the *purity* of a clustering to formalize this intuition. In the case where we have no labeled data (where root cause diagnosis is known), we can distinguish signatures only in terms of their corresponding SLO state (compliance or violation).

If given labeled data, we can count the number of signatures in each cluster with each label. These counts are then normalized by the total number of signatures in each cluster (sum of counts) to produce probability estimates $p_1, \ldots, p_n$ if there are n different problem labels, where $\sum_{i=1}^{n} p_i = 1$. These are used in turn to score the purity of each cluster. A cluster is *pure* if it contains signatures of only one type of problem (as determined by the labels), i.e., if exactly one of $p_1, \ldots, p_n$ equals 1 while the other probabilities equal 0. With these probabilities, we can compute the *information entropy* (or just entropy) of each cluster, given as: $H = \sum_{i=1}^{n} -p_i log_2(p_i)$. For a pure clustering, entropy is 0, which is the best possible result. The entropy is 1 when a cluster is evenly split between each label ($p_1 = \ldots = p_n = \frac{1}{n}$). When labeled data is unavailable, we may only consider the percentage of violation signatures p_v versus compliance signatures p_c in a cluster and use those probabilities in the entropy formula by letting $n = 2$, $p_1 = p_v$, and $p_2 = p_c$.

We compute the overall average entropy of all of the clusters weighted by the normalized cluster size to give us a measure of purity of the entire clustering result. Average clustering entropy being close to 0 would be a strong indication that the signatures capture meaningful characteristics for distinguishing different problem types (if evaluated using labeled violations), or at least meaningful characteristics of violations in contrast to periods of non-violations (if evaluated on unlabeled data).

The k-means algorithm must be given the parameter k, the total number of clusters expected. There are a number of procedures for determining the optimal parameter settings, including score metrics with regularization components and search procedure which increase their value gradually until no significant improvement in the clustering distortion is achieved [18]. We choose a simpler approach and merely iterate through values of k obtain clustering purity results for each iteration. For clustering evaluation where SLO violations are labeled with their problem type, we expect that as long as k is set to be greater than n (the number of different problem labels), overall clustering entropy should be close to 0. However, this is not generally true when clustering violations and non-violations together (given the lack of labeled data), although entropy is still expected to generally decrease as k increases.

3.5 Summary

This chapter explored signature usage and evaluation methodology.

- Once signatures are produced, they may be used for retrieval or for clustering. Both usages depend on using a formal distance metric to represent similarity between signatures. We use a common vector distance metric known as the *cityblock* or L_1 *norm* distance. The hope is that signatures closer in distance are more likely to represent the same system problem.

- Signature retrieval attempts to find signatures that are closest to some *test signature*. We can evaluate retrieval quality using a *precision-recall curve*, which is commonly employed in the information retrieval domain. Evaluation in this manner is only possible when signatures are properly *labeled* (or *annotated*) according to root cause information.

- Clustering aims to identify natural groupings of signatures that characterize different system problems. In addition to the distance metrics, a clustering method that seeks to minimize distortion is needed. We use the popular k-means clustering algorithm. Clustering quality can be evaluated using a purity score. This score is based on the ratios of different types of problems (as determined by each signature's label) in each cluster. When labels are unavailable, clustering evaluation may be solely based on the ratio of signatures representing SLO violation epochs versus those representing SLO compliance epochs in each cluster.

Chapter 4

Constructing signatures

We now present a method for constructing system signatures using *metric attribution*, a statistical technique introduced by Cohen *et al.* in [15]. A key insight of this approach is that relevant information about the cause and effect of a system problem are usually adequate to *define* the problem and *differentiate* it from different types of problems. Therefore, identifying the subset of relevant information and incorporating that into signatures should allow these signatures to be effectively used for clustering and retrieval, the main subjects of the preceding chapter. This chapter first presents a sketch of our approach and then describes each step in more detail.

4.1 Sketch of the approach

The process of signature construction starts with the collection of system information. Software such as HP's OpenView or the open source Ganglia is usually used. Although the collection and reporting of data for large system is not trivial, the exploration of this aspect of the process is outside the scope of this thesis. For our approach, this system information must include a high level indicator of performance in the form of an *service level objective* (SLO), explained in Section 2.2.2. It must also include low level metric data, such as CPU and memory utilization, that may be related to high level system behavior. We focus on three-tiered Internet services because these types of information are already typically captured in production systems.

As this system data is collected, we induce Bayesian network classifiers, Tree-Augmented Naive Bayes or TAN in particular (see Section 2.3.2), to predict high level system behavior (represented by SLO state) from the low level metric data. We evaluate the prediction accuracy of these models to determine how well they capture meaningful patterns in the system data. Since Bayesian network classifiers have the key advantage of being interpretable, we can interrogate the models to find out exactly which low level metrics most strongly influenced their classification decisions. We deem these low level metrics as *attributable* to high level behavior and use this information as the basis for system signatures.

4.2 Inducing models on system data

The behavior of even relatively simple networked systems are quite difficult to model completely, even with detailed *a priori* information about the configuration and make up of the system. Therefore, it is foolish to expect to be able to completely model the behavior of a complex system by using statistical machine learning techniques on system measurements. Fortunately, we do not need to fully model a system's behavior. By using the simple delineation of desirable versus undesirable system states that SLOs provide us, we can transform an almost impossibly complex modeling task into the relatively simple task of binary classification.

4.2.1 Formalizing the problem

Given the background of Chapter 2, we can now cast the problem of modeling system behavior as a classification task in supervised learning. Let $Y_t \in \{s^-, s^+\}$ denote whether the system is in compliance (s^-) or noncompliance (violation) (s^+) with the SLO at time epoch t[1], which can be measured directly. Let $\vec{M}_t$ denote a vector of values for n collected metrics $[m_0,\ldots,m_n]$ at time t (the subindex t will be omitted when the context is clear). The classification task here is to induce or learn a classifier function $\mathcal{F} : \vec{M}_t \rightarrow \{s^-, s^+\}$ that maps the universe of possible values for $\vec{M}_t$ to one

[1]The length of an epoch is not in any way restricted by our approach. However, due to the performance impact of monitoring processes, epochs are usually in the one to five minute range.

of two states $\{s^-, s^+\}$. The training data set for this task consists of observations of the form $< \vec{M}_t, Y_t >$ collected from a system in operation.

To evaluate the success of our classifier function $\mathcal{F}$, we could determine the classification accuracy, which in this case is defined as the probably that $\mathcal{F}$ correctly identifies the SLO state S_t associated with any $\vec{M}_t$. However, this measure can be misleading if the test set contains predominantly data belonging to one class. We expect this to be the case in any reasonably well-designed production system since SLO compliance or non-violations s^- should be much more frequently observed than SLO violations s^+. To see why raw classification accuracy may be misleading, imagine that only 10% of epochs are SLO violations. In that case, a trivial classifier that always predicts compliance yields a classification accuracy of 90%. Therefore, our figure of merit is *balanced accuracy* (BA), which averages the probabilities of correctly identifying non-violations and violations. Formally:

$$BA = \frac{P(s^- = \mathcal{F}(\vec{M})|s^-) + P(s^+ = \mathcal{F}(\vec{M})|s^+)}{2} \tag{4.1}$$

To achieve the maximal BA of 100%, $\mathcal{F}$ must perfectly classify both SLO violation and SLO compliance incidences. The trivial classifier in the previous example would only achieve a BA of 50%. Note that since BA scores the accuracy with which a set of metrics $\vec{M}$ predicts the SLO state, it is an indicator of the degree of correlations between these metrics and higher level system behavior.

Unlike traditional usage of classifier models, our goal is not to simply use an induced model to predict the SLO state Y_t for some $\vec{M}_t$ where the SLO state is not already known. This is because Y_t is constantly being observed. There is never a need to identify the SLO state from the values of individual metrics. Rather, the goal of this technique is to induce a classifier model and then interrogate them for the patterns and correlations that they have captured. The predictive value of the model merely serves as an indictor of the strength of the patterns. To achieve the goal of extraction, the classifier model must be interpretable, which precludes many common classification algorithms. Therefore, we use an interpretable class of models based on Bayesian networks.

To implement $\mathcal{F}$, we use a TAN (other types of Bayesian networks could also be used) to represent the joint distribution $P(Y, \vec{M})$ – the distribution of probabilities for the system state and the observed value of the metrics. From the joint distribution we compute the conditional distribution $P(Y|\vec{M})$ and the classifier uses this distribution to evaluate whether $P(s^+|\vec{M}) > P(s^-|\vec{M})$. If so, $\mathcal{F}$ would predict s^+, SLO violation. Using the joint distribution enables us to invert $\mathcal{F}$, so that the influence of each metric on the prediction of SLO state can be quantified with sound probabilistic semantics. This is the basis for metric attribution and will be derived formally in 4.3.

Although the joint distribution is represented in a Bayesian network, the probability distribution of the individual metrics must still be chosen. The simplest method is to assume a *Gaussian* distribution (also known as a *normal* distribution). Two distributions are assumed for each metric, one for each possible SLO state. By using the Gaussian distribution, inducing a model is very simple and involves only calculating the means and variances for each metric for each of the two SLO states. Such a model also uses very little memory. However, not all metrics can be accurately represented using Gaussian distributions. The alternatives and their implications will be discussed later in Chapter 8.

4.2.2 Feature selection

Armed with monitored low level system metric data $\vec{M}$ and high level SLO state Y, we could simply learn a classification function $\mathcal{F} : \vec{M} \rightarrow Y$ as described in the previous section. However, such an approach is unlikely to produce an accurate classifier. The reason is known as the dimensionality problem in pattern recognition and statistical induction: the number of data samples needed to induce *good* models increases exponentially with the dimension of the problem, which in this case is the number of metrics in the model (which influences the number of parameters). The solution to this problem is usually a process called *feature selection*, where only the subset of metrics most relevant to modeling the patterns and relations in the data is retained.

The feature selection process provides two advantages. First, it discards metrics that appear to have little impact on SLO state, allowing human operators to focus on a smaller set of candidates for diagnosis. Second, it reduces the number of data samples needed to induce robust models, due to the aforementioned dimensionality problem. At first glance, it may seem that the role of metric attribution and feature selection overlap somewhat. They both seek to determine which are the relevant metrics with respect to SLO behavior. However, feature selection is often a very computationally intensive task that is only done once for an entire data set, as a preprocessing step, while metric attribution requires only a few probability calculations and determines which metrics of a model are relevant for *each epoch*. If a metric is deemed to be irrelevant during the feature selection process, it will never be considered by the classifier model and thus will never be attributed for any SLO violation. Conversely, if a metric is considered by the classifier model, it could be considered attributed for none, some, or all SLO violations.

The problem of feature selection is essentially that we wish to select $s\vec{M}$, a subset of $\vec{M}$, that produces the most accurate classifier function $\mathcal{F} : s\vec{M} \rightarrow Y$. This means that if $\vec{M} = [m_1, \ldots, m_n]$, there are a total of 2^n different subsets $s\vec{M}$, and it is essentially a combinatorial optimization problem that is usually solved using some sort of heuristic search. The method used in [15] is a greedy search approach as follows. $s\vec{M}$ is initially empty. Iterate over all of the metrics $m_1, \ldots, m_n$ and choose m_{x1} such that $\mathcal{F} : s\vec{M} = [m_{x1}] \rightarrow S$ has the best accuracy over the dataset[2]. Once m_{x1} is chosen, we again iterate over $m_1, \ldots, m_n$ but not including m_{x1}. We choose m_{x2} such that $\mathcal{F} : s\vec{M} = [m_{x1}, m_{x2}] \rightarrow S$ has the best accuracy. We continue this process until accuracy no longer improves by adding more metrics or when a fixed upper limit is reached.

Although feature selection is very successful at mitigating the dimensionality issue and helping to produce robust models, it does introduce one significant problem. Metrics that are removed from consideration due to lack of relevance will *never* be considered. This was acceptable in the context of [15] because the evaluation data consists of monitored system metrics and SLO state over the course of several hours.

[2]Ten-fold cross validation [28] is used to prevent *overfitting*, a common problem for statistical approaches

It was assumed that the behavior patterns and correlations in each data set never changed. However, the longer the time period, the less acceptable the assumption. For this technique to be useful in a real system, it needs to be able to adapt to changes in system behavior. The necessary step of feature selection complicates this ability to adapt. This challenge and how we resolve it will be described in detail in Chapter 6.

4.3 Metric attribution

Metric attribution is the statistical process by which we identify low level system metrics that are the most correlated to high level behavior. The metrics that are correlated are referred to as *attributed metrics*. This section will review the original technique introduced by Cohen, *et al.* in 2004 [15]. Extensions of that process are summarized in the next section and explored in detail in later chapters.

Given a model induced as described in section 4.2.1, the joint distribution that it represents can be expressed in a functional form as a sum of terms, each involving the probability that the value of some metric m_i occurs in each state (s^- or s^+) given the value of the parent metric (if any) m_{p_i} on which m_i depends (in the probabilistic model):

$$\sum_{i=1}^{n} \log[\frac{P(m_i|m_{p_i}, s^+)}{P(m_i|m_{p_i}, s^-)}] + \log\frac{P(s^+)}{P(s^-)} > 0 \tag{4.2}$$

From Eq. 4.2, a metric m_i is implicated in an SLO violation if $log[\frac{P(m_i|m_{p_i},s^+)}{P(m_i|m_{p_i},s^-)}] > 0$, also known as the log-likelihood ratio for metric m_i. To analyze which metrics are implicated with an SLO violation, we simply examine which metrics had a positive log-likelihood ratio. These metrics are flagged as *attributed* for that particular violation. Furthermore, the *strength* of each metric's influence on the classifier's choice is represented by the value of the log-likelihood difference.

It is the hope that attributed metrics can point to the root cause of SLO violations and that signatures based on this attribution information can define and differentiate system problems. Unfortunately, as with any statistical technique, the patterns that are captured by models could be mostly noise, in which case attributed metrics may not be any more relevant than those that are not attributed. In addition, the attribution of a metric only means that the metric is *correlated* with the overall SLO state. It is not evidence of a *causal* link. Fortunately, signatures can be effective even if attribution information fails to completely capture true root cause since the set of metrics that are *likely* relevant to a problem is usually enough to differentiate it from other problems. True root cause diagnosis is not possible without *a priori* system specific knowledge anyway.

4.4 Extensions to original metric attribution procedure

To produce accurate signatures, several changes to the original attribution procedure were necessary:

- In order to adapt to the ever changing behavior patterns of systems, new statistical models are induced periodically and added to an *ensemble* of models. The attribution value of a metric is calculated using all relevant models in the ensemble. The insight of this approach is that we can model complex and constantly evolving system behavior by using a collection of simple models designed to represent a single behavior. This extension is described in Chapter 6.

- Just as a system can misbehave in a wide variety of ways, *good* (SLO compliant) system behavior can also vary. The same system problem can appear different when compared to different good system states. Therefore, all problem behaviors must be compared to the same "*baseline*" good behavior or behaviors. However, choosing behaviors to use as the baseline is not trivial. This issue is covered in Chapter 7.

- A problem is better characterized by all strong correlations between that problem state and the system metrics rather than by only the strongest correlation. Each classifier may be thought of as representing a single correlation and the accuracy of the classifier as strength of the correlation. Therefore, it is better to utilize all models that pass some minimum accuracy threshold, rather than searching for the most accurate model only.

4.5 Signature based on metric attribution

The process of using metric attribution information to construct system signatures is fairly simple. The ultimate goal is to transform observations of the form $< \vec{M_t}, Y_t >$ into $\vec{S_t}$ for each epoch t (e.g. five minute intervals). Recall that we start by inducing classifier models on the system observations and then compute metric attribution information using those models. We define $\vec{S_t} = [a_1, \ldots, a_n]$ where $[a_1, \ldots, a_n]$ represents the attribution value of the metrics $\vec{M_t} = [m_1, \ldots, m_n]$. Each of $a_i (i = 1, \ldots, n)$ can take on one of three attribution values, the meanings of which are described as follows:

- a_i = 1 if the metric m_i is generally correlated[3] to overall SLO state and the current value of m_i strongly indicates a state of SLO violation[4]. We say the metric is *attributed*. (see Figure 4.1)

- a_i = -1 if the metric m_i is generally correlated to overall SLO state but the current value of m_i strongly indicates a state of SLO compliance or non-violation[5]. We say the metric is *inversely attributed*. Note: this is rare for signatures of epochs when the SLO is violated. (see Figure 4.1)

- a_i = 0 if the metric m_i is not generally correlated to overall SLO state, or if it is correlated but the current value of m_i does not strongly indicate either of the possible SLO states. Such a metric is referred to as *not attributed*. (see Figure 4.2)

[3]Generally correlated refers to being considered by a model. In order to be considered by a model, a metric has to first pass the feature selection process, which eliminates metrics with no correlations to overall SLO state.

[4]Log-likelihood ratio ≥ 5, as calculated by the formula given in 4.3. The specific threshold is configurable although it must be positive.

[5]Log-likelihood ratio ≤ -5, as calculated by the formula given in Section 4.3. The specific threshold is configurable although it must be negative.

Note that the attribution value of a metric during an epoch does not directly depend on the SLO state of that epoch.

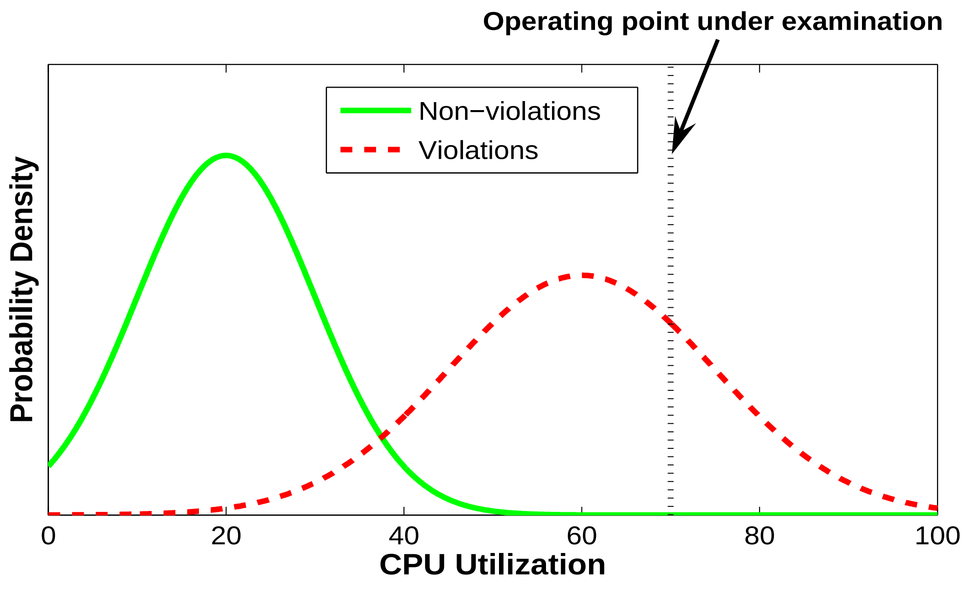
The signature vector $\vec{S_t}$ for each time epoch can be added to a database as they are created. As problems in the system occur, this database of signatures can be queried in various ways to aid operator diagnosis efforts. In addition, signatures can be manually annotated with information about root causes or possible resolution strategies once they are found.

4.6 Other signature compositions

Although we have described signature construction only in terms of metric attribution, there is nothing inherent about the concept of signatures and their usage model that depends on utilizing only attribution information. In fact, signatures could be constructed without using metric attribution information at all. The only issue is the effectiveness of those signatures in achieving the goals we set out for our approach. The following are some alternates to metric attribution that could be used alone, combined with one another, or combined with metric attribution.

- **Raw metric values.** Directly using raw metric values works poorly as a signature because the range of each metric can very substantially (compared to other metrics). Following common practice in data analysis [18], each metric should have its values *normalized* to $[0, 1]$, using the range of values ever seen, to prevent scaling issues from influencing similarity metrics and clustering. Future references to raw values will assume normalization has taken place already. Raw metric values can also be multiplied by the metric attribution values to roughly represent a signature based on "only raw values of metrics correlated to SLO state."

- **Loglikelihood ratio.** Since metric attributions values are calculated using the Loglikelihood ratio, it is certainly a conceivable option to use this ratio directly. Metrics with values that more strongly indicate SLO violation would be represented by higher positive numbers, while those that more strongly indicate



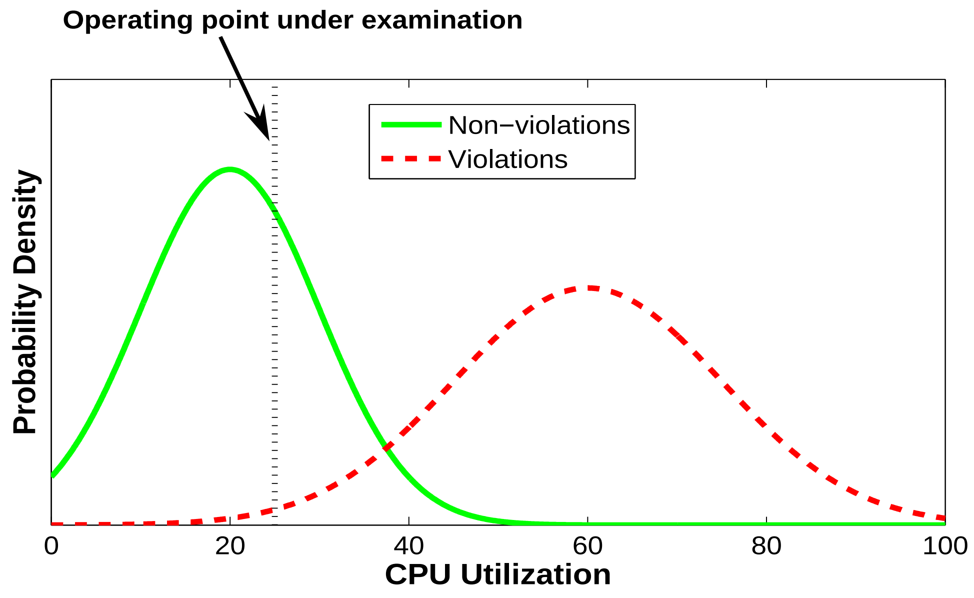
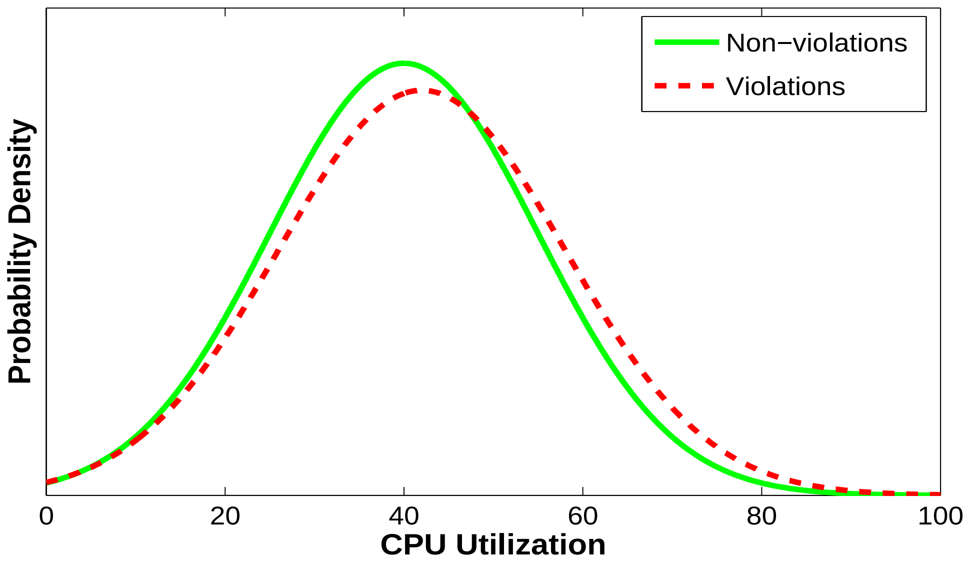


Figure 4.1: Example of a metric considered *attributed*(**top**) and *inversely attributed*(**bottom**). A previously induced Bayesian classifier has modeled CPU Utilization as two Gaussian distributions, one for each SLO state. The distribution for SLO violation a mean of 60% and standard deviation of 15%, while for SLO compliance the mean is 20% with a standard deviation of 10%. A CPU Utilization at 70% for some epoch would strongly indicate a state of SLO violation and thus this metric would be considered *attributed*. Conversely, CPU Utilization at 25% would strongly indicate SLO compliance and thus the metric would be considered *inversely attributed*

.



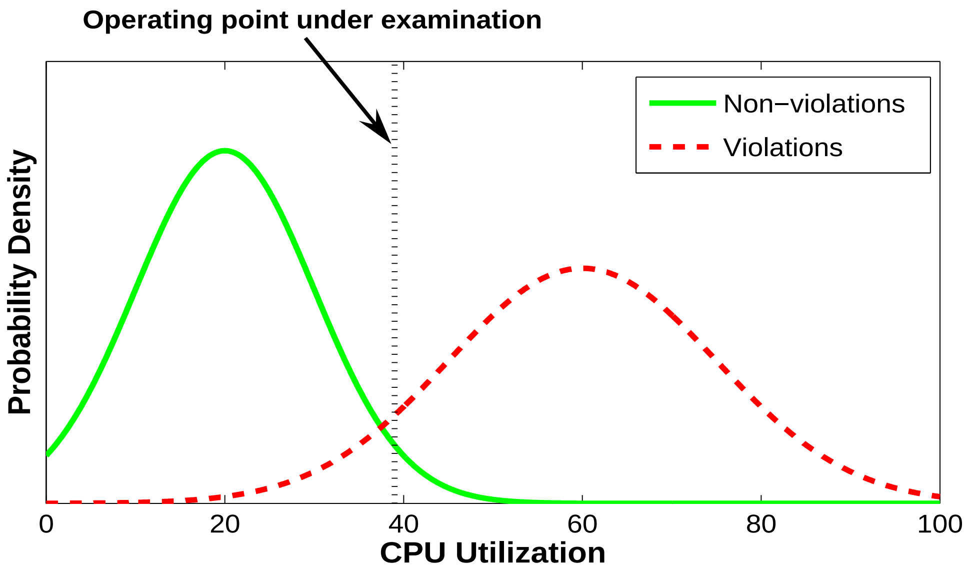


Figure 4.2: Examples of a metric considered *not attributed*. **(top)** The CPU Utilization metric's SLO non-violation and violation distributions highly overlap. This suggests a lack of correlation between CPU Utilization and overall SLO state. Therefore, this metric would be removed from consideration during the feature selection process. **(bottom)** Although the distributions in this case indicate correlation with SLO state, the value of CPU Utilization during a particular epoch (39% in this case) may not strongly point to SLO violation or compliance.

SLO compliance are represented by more negative numbers. The issue with using this ratio directly is that it is not bounded and not calibrated[6].

- **Relative value.** While metric attribution can tell us if a metric is correlated to overall SLO state, it says nothing about the type of correlation. It answers the question of if a higher value for the metric indicates a greater or smaller likelihood for SLO violation? (1 if yes, -1 if no)

- **Relevance score.** The relevance score of a metric represents the balanced accuracy of the classifier that uses the metric. A higher relevance score usually indicates a stronger correlation to SLO state. The range of this score is 0 to 1.

Other factors not included above could also be used. It would be impractical to evaluate every conceivable signature compositions. However, we did evaluate several compositions besides using metric attribution by itself and found using attribution only was almost always superior for our evaluation data sets. Empirical results can be found in Chapter 5. This does not imply that using different compositions would never be helpful for any system. Therefore, we recommend using metric attribution only signatures as a default setting and allowing operators to experiment with other compositions to determine if those are better suited for their system.

4.7 Summary

This chapter explained a specific technique for constructing signatures by utilizing a statistical technique known as *metric attribution*. The process of building signatures in this manner is as follows.

- Induce TAN (a class of Bayesian network classifier) models on observed system data (low level metrics plus high level SLO state).

- Extract metric attribution information from these TAN models using the procedure described in Section 4.3. Although the original technique is based on work

[6]Not being calibrated implies that the value is useful only as a comparator between ratios generated by the same statistical model.

in [15], several improvements to the process were necessary and are described in later chapters. These changes relate to the adaptivity, robustness, and scope of attributions.

- Each signature is simply a vector of attribution values, one for each low level metric. For any epoch, each metric may be considered *attributed*, *inversely attributed*, or *not attributed*. Although we focus on attribution based signatures, other compositions are possible. We will compare the results of using different compositions in the next chapter.

The next chapter will show the results of evaluating metric attribution based signatures for retrieval and clustering, while Chapter 6 and Chapter 7 will describe extensions to metric attribution for adapting to system changes and for dealing with variability in SLO compliant behaviors, respectively. Chapter 8 presents a practical guide for operators seeking to utilize our approach.

Chapter 5

Evaluation datasets and results

This chapter describes the datasets that we use to evaluate our approach as well as the results of the evaluation. We infer the effectiveness of signatures by how well the two main usage methods, clustering and retrieval, perform on data collected from systems in operation. Our evaluation criteria are defined operationally, e.g., to say that retrieval "performs well" is to say that the signatures retrieved as being similar to some problem do indeed represent problems with similar root-cause diagnosis in practice. Furthermore, to say that signatures are accurate, meaningful, and of high quality is to imply that retrieval and clustering based on those signatures perform well.

We first introduce the systems from which we collected our data sets and the known problems in each system. We then present the results of evaluating signature retrieval and clustering using these data sets and also discuss anecdotal evidence about leveraging signatures across different systems. Finally, the performance impact of our approach and the limits of this evaluation are discussed. Note that each problem examined is performance-based. This is due to the SLOs of these systems being based on a performance metric (average transaction response time). Our approach does not restrict us to any particular class of problems.

5.1 Trace collection

Our empirical results are based on large and detailed traces collected from two distributed applications, one serving synthetic workloads in a controlled laboratory environment and the other serving real customers in a globally-distributed production environment. These two traces allow us to evaluate our approach in complementary ways. The testbed trace is labeled with known root causes of problems. The causes are known because we controlled the faults injected into the system. Labeled data allow for full evaluation of our signature-based diagnostic methods using common information-retrieval performance measures. The production trace is not labeled as reliably[1] or as thoroughly as the testbed trace. However, we may apply the clustering technique for unlabeled data described in section 3.4. In addition, the presence of some labels allows us to further validate the accuracy of our retrieval on this real-world data.

Our traces record two kinds of data about each service: application-level performance data to use for deriving system SLO state, and low level system resource utilization metrics (e.g., CPU utilization). Our tools generally capture the latter as averages in non-overlapping windows (epochs).

5.2 Experimental testbed traces

5.2.1 System architecture

Our controlled experiments use the popular PetStore e-commerce sample application, which implements an electronic storefront with browsing, shopping cart and checkout. Each tier (Web, J2EE, database) runs on a separate HP NetServer LPr server (500 MHz Pentium II, 514 MB RAM, 9 GB disk, 100 Mbps network cards, Windows 2000 Server SP4) connected by a switched 100 Mbps full-duplex network (see Figure 5.1). Apache's extended HTTPd log format provides us with per-transaction response times and we obtain system level metrics from HP OpenView Operations Agents running

[1]That is to say, we are less confident about the accuracy of the labels that we do have for this data set. We discuss this limitation in section 5.7.

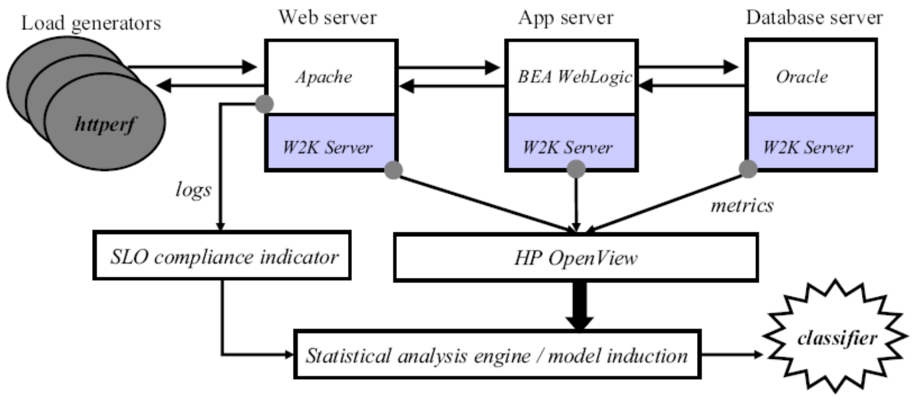


Figure 5.1: **Our experimental testbed system, featuring a commonly used three-tier internet service.**

on each host. A detailed description of our testbed's hardware, software, networking, and workload generation is available in [54]. We collected 62 individual metrics at 15-second intervals and aggregate them into one-minute windows containing their means and variances. We pre-process our raw measurements from the Apache logs to average transaction response times over the same windows and then join all data from the same application into a single trace for subsequent analysis.

We use the standard load generator `httperf` [38] to generate workloads in which simulated clients enter the site, browse merchandise, add items to a shopping cart, and checkout, with tunable probabilities governing the transition from "browse item" to "add item to cart" (probability P_b) and from "add item to cart" to "checkout cart" (probability P_c). We measure the average response time of client requests in each 1 minute window and require that the average response time stay below 100 msec to maintain SLO compliance.

5.2.2 Inducing SLO violations

We created three handcrafted fault loads designed to cause SLO violations. In the first, we alternate one-hour periods of $P_b = P_c = 0.7$ with one-hour periods of

$P_b = P_c = 1.0$. This problem was termed `BUYSPREE`. In the second, we execute a parasitic program on the database server machine that consumes approximately 30% of available CPU cycles during alternating one hour intervals, but no other major resources. The last faultload does the same thing but on the application server rather than the database server. These two scenarios were called `DBCPU` and `APPCPU`, respectively.

Note that these faultloads simulate both SLO violations due to internal problems (CPU contention) as well as problems resulting from change in workload (extreme buying patterns in `BUYSPREE`). In both cases, the injected failures correspond to the root causes of performance problems, which we use as the ground truth for proper labeling of signatures of SLO violations.

5.3 Production system traces

5.3.1 System architecture

Our second trace is based on measurements collected at several key points in a globally-distributed application that we call "FT" for confidentiality reasons. FT serves business-critical customers on six continents 24 hours per day, 365 days per year. Its system architecture therefore incorporates redundancy and failover features both locally and globally, as shown in Figure 5.2. Table 5.1 summarizes key hardware and software components in FT, and the transaction volumes recorded by our traces (Table 5.2) demonstrate the non-trivial workloads of the FT installations. All hosts at the application server and database server tiers are HP 9000/800 servers running the HP-UX B.11.11 operating system, except that one database server in Asia is an HP rp7410 server.

HP OpenView Performance Agent (OVPA) provides system utilization metrics for application server and database hosts. FT is instrumented at the application level with Application Response Measurement (ARM) [50], providing transaction response times. OVPA and ARM data area aggregated into 5-minute epochs in the processed traces we analyze. We have traces from the Americas and Asia/Pacific hubs but not

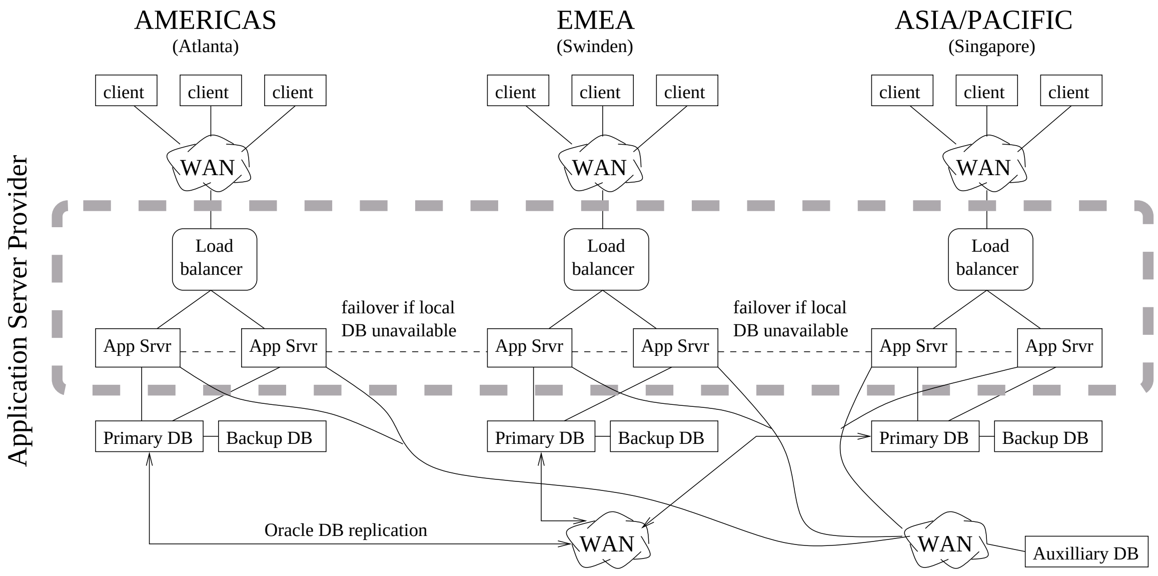


Figure 5.2: **Architecture of the "FT" production system.** FT is a globally-distributed multi-tiered application with regional hubs in the Americas, Europe/Middle East/Africa, and Asia. Different organizations are responsible for the FT application and the application server on which it runs; the latter is indicated by a shaded dashed rectangle in the figure. FT has a globally-distributed main database and an additional auxiliary database, managed by a third organization, shown in the lower right.

Region	Role	# hosts	# CPUs	# disks	RAM (GB)
Amer	App srvr	2	16	16	64
Amer	DB srvr	2	12	18	32
EMEA	App srvr	3	16	10	32
EMEA	DB srvr	2	6	?	16
Asia	App srvr	2	12	63/22	20
Asia	DB srvr	2	6	8	16

Table 5.1: **Key hardware and software components in FT.** The two app server hosts in Asia have different numbers of disks. All app servers ran WebLogic and all DB servers ran Oracle 9i. Most of the DB servers had 550 MHz CPUs.

		transactions/min			% SLO
Server	Dates	mean	95 %	max	viol
AM1	12/14–1/14	208.6	456.4	1,387.2	23.6
AM2	12/13–2/08	207.9	458.0	977.4	22.5
Asia1	12/17–1/05	39.9	118.2	458.4	26.2
Asia2	12/17–1/30	52.1	172.8	775.0	13.1

Table 5.2: **Summary of FT application traces.** The last column is the percentage of transactions which violated their SLO in the data. Trace collection began in late 2004 and ended in early 2005. "AM" and "Asia" servers were located in the Americas and Asia, respectively.

from Europe. Table 5.2 summarizes our FT traces. Our criterion for SLO violation is whether the average response time over all transactions in a 5-minute period exceeded 4 seconds.

FT is well suited to our interests because its requirements include high performance as well as high availability. Performance debugging in FT is particularly challenging for two reasons. First, different organizations are responsible for FT itself and for the application server infrastructure in which crucial FT components run (the latter is delimited with a shaded dashed oval in Figure 5.2); opportunities for inter-organizational finger-pointing abound when end-to-end performance is poor. Second, FT's supporting infrastructure is physically partitioned into three regions with separate operational staffs. A performance problem that occurs in the afternoon in each region, for example, will occur three different times, will appear to be specific to a single region each time it occurs, and will demand the attention of three separate teams of system operators. Cost-effective diagnosis in FT requires that commonalities be recognized across regions, across time, and across organizational boundaries, and that different teams of human diagnosticians leverage one another's efforts.

Another attractive feature of our FT traces is that some of our 5-minute samples are labeled in that they correspond to times when we know that a specific performance problem occurred whose root cause was subsequently diagnosed. The next section will describe this particular problem, which illustrates both the challenges that we face and the opportunities that our approach attempts to exploit.

5.3.2 Diagnosed problem: Insufficient Database Connections (IDC)

The FT production system experienced a recurrent problem mainly in the Americas domain during December 2004 and January 2005. During episodes of this problem, business-critical customers experienced latencies of several minutes on transactions that normally complete within seconds. The operators who first detected the problem described it as “stuck threads” in the application server because WebLogic issued messages in a log file each time it diagnosed a stuck thread. Since there can be many causes for threads to become stuck, it was necessary to look for other symptoms to diagnose the cause.

Due to the severity of the problem, a joint task force comprising both FT application developers and application server administrators quickly formed to address it. This team eventually diagnosed and repaired the root cause of the performance problem, thus providing labels for data points in our traces corresponding to episodes. Our account of the problem is based on detailed bug-tracking database entries and e-mail correspondence among the troubleshooters.

After several weeks of debugging, the problem was traced to an insufficient pool of database connections. Under heavy load, application threads sometimes had to wait more than 10 minutes to acquire a connection, and were therefore flagged as “stuck threads” by WebLogic. The problem was solved by increasing the connection pool size by 25%. We use the label `IDC` (Insufficient Database Connections) to refer to this problem from now on.

5.4 Signature compositions

Our evaluation compares four different approaches for creating signatures (examples in Table 5.3). See Chapter 4 for details on signature construction methods. Let $\vec{S}$ denote the vector representing the signatures. In all cases the elements s_i in this vector correspond to a specific system, application, or workload metric.

1. Raw values: in this case we represent a signature $\vec{S}$ using the normalized raw values of the metrics, as described in Section 4.6. This signature is the most naive and requires no extra processing of the traces.

2. Metric attribution: Following the attribution definitions provided in Section 4.5, if metric m_i is deemed attributable, then $s_i = 1$. If it is inversely attributed, $s_i = -1$, and $s_i = 0$ for all other cases. Although this requires significant computation [15], individual attribution values can be determined on the millisecond timescales, which allows this approach to used in real time. (see Section 5.6)

3. Metric attribution and raw values: This is similar to the previous approach except that the raw value of the metric is multiplied by its metric attribution value as explained in the previous item. The intuition here is that the information contained in the value of the metric is added to the information in the attribution process.

4. Loglikelihood ratio: This is the ratio $log[\frac{P(m_i|m_{p_i},s^+)}{P(m_i|m_{p_i},s^-)}]$ (see Section 4.3) that is used in computing the final attribution value. For this approach, we directly use the value of this ratio in the signature.

5.5 Results

This section presents and discusses the results of evaluating our approach with our two traces. We will use `TESTBED` to identify the trace from the experimental testbed and `FT-TRACE` for the data from the globally-distributed production environment system. For the `TESTBED` experiments, we intentionally injected known faults into the system to cause SLO violations. The `TESTBED` trace is therefore reliably *labeled* with the appropriate diagnosis per epoch. `FT-TRACE` is only partially labeled with the one diagnosed problem described in section 5.3.2. These labels, as well as the objective measure of system state reflected by SLO compliance or violation, provide us the ground truth against which we will determine the effectiveness of our signatures with respect to clustering and retrieval.

Metric Name	Raw Value	Metric Attr	Raw Value & Attr	Loglikehood Ratio
transaction count	398.00	0	0	0.4
gbl app cpu util	97.47	1	97.47	15.4
gbl app alive proc	449	0	0	-0.6
gbl app active proc	357	0	0	1.2
gbl app run queue	10.57	1	10.57	6.7
gbl app net in packet rate	817	1	817	275.4
gbl mem util	54.62	1	54.62	25.9
gbl mem user util	26.32	1	26.32	10.2
DB1 CPU util	25.96	-1	-25.96	-33.7

Table 5.3: **Examples of the different signature compositions**, showing a subset of the metrics collected in the production environment. The first column is of raw values (not normalized to preserve the context of these metrics), second is metric attribution (with possible values in $\{+1, 0, -1\}$), third is the product of raw values and metric attribution, and the last column is the loglikelihood ratio used in computing attribution.

5.5.1 Retrieval

We now evaluate the different signature compositions based on the quality of retrieval operations and demonstrate that signatures based on metric attribution are superior for our purposes. This strongly implies that metric attribution captures information about system state that goes beyond the raw values of the collected metrics, further validating the results from [15].

We also remark that in our experiments we saw that certain metrics are consistently deemed irrelevant for all time epochs in the traces (e.g., in the `FT-TRACE` data the root CPU, memory, and disk utilization on the application server were consistently eliminated during the feature selection process, resulting in $s_i = 0$ for all epochs). In some cases such an observation can lead to reducing the number of metrics being collected, although that loss of information can be detrimental when a dropped metrics becomes relevant in future problems. In addition, because our modeling process is able to quickly narrow down the number of metrics considered to only those that show correlations with overall SLO state, unless the expense of data collection is significant, we discourage removing any metrics from the measurement apparatus.

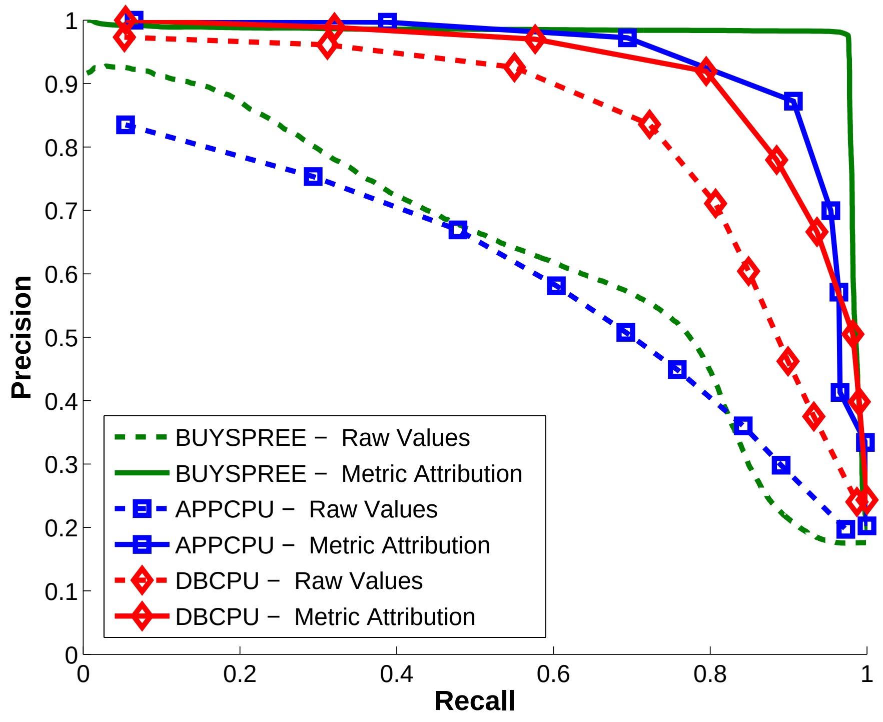


Figure 5.3: **Precision-recall curves for TESTBED traces.** Precision-recall representation of retrieval behavior for metric attribution and raw value based signatures. In general, the larger the area under the curve the better. An ideal PR curve has precision equal to 1 for all values of recall.

Figure 5.3 shows the precision-recall curves of retrieval exercises performed on the `TESTBED` traces using the metric attribution versus raw values signature compositions (see Section 3.3 for the definition of these curves). Precision-recall performance is better in proportion to the area under its curve. Clearly, the use of attribution information provides an advantage, as the curves using attribution are far superior to using raw values alone. The area under the PR curve is compared in Table 5.4 for all four signature compositions.

Our real-world traces were collected during a period when a misconfiguration was causing a performance problem (the `IDC` problem discussed in Section 5.3.2). Overall, 318 epochs were labeled with this problem. Figure 5.4 shows the location of these epochs over the month long trace, overlaid on the value of the reference

Testbed Trace	Raw Value	Metric Attr	Raw Value & Attr	Loglikehood Ratio
APPCPU	0.59	0.94	0.88	0.77
DBCPU	0.83	0.92	0.92	0.64
BUYSPREE	0.63	0.97	0.94	0.64

Table 5.4: Area under precision-recall curves for testbed traces. The larger the area under a PR curve the better, with an ideal curve having an area equal to 1.

metric (average transaction response time). It can be seen that this issue occurred intermittently over that period.

Figure 5.5 shows the precision-recall graphs for retrieving signatures of the `IDC` problem. In the `TESTBED` trace, metric attribution based signatures again performed the best. The graph shows that high precision is achieved for a wide range of the recall value. For example, for a retrieval set of the 100 signatures closest to a test signature (where that test signature has been labeled with `IDC`), an average of 97 of those were also labeled `IDC` and thus correctly retrieved. This corresponds to a precision of 97% with a recall of 30.6%. Such a precision would be more than sufficient for an operator to safely infer that the label attached to the majority of signatures being retrieved matches the problem described by the test signature. These results confirm that the use of attribution information in the generation of signatures allows effective inference of the similarity of system problems.

5.5.2 Clustering

Recall that clustering is useful to operators when many problems are undiagnosed (i.e. many signatures are unlabeled). It provides information about the intensity and recurrence of all system problems and gives troubleshooters an educated way of prioritizing diagnostic efforts. In addition, in the absence of fully labeled data, such as the case for `FT-TRACE`, clustering results may still be used for evaluation of our approach.

We cluster signatures as described in Section 3.4. To evaluate the quality of the clustering, we utilize the notion of purity introduced in that same section. Entropy is used as a measure of purity with low entropy implying that each cluster contain

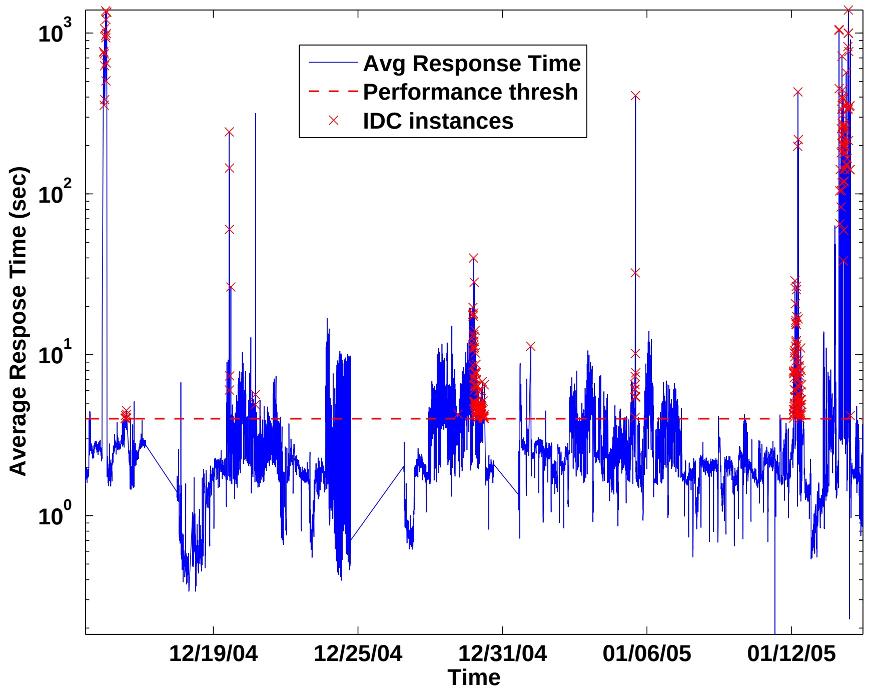


Figure 5.4: **Temporal location of the instances of the IDC problem on one of the Americas machines, overlaid on the reference metric.**

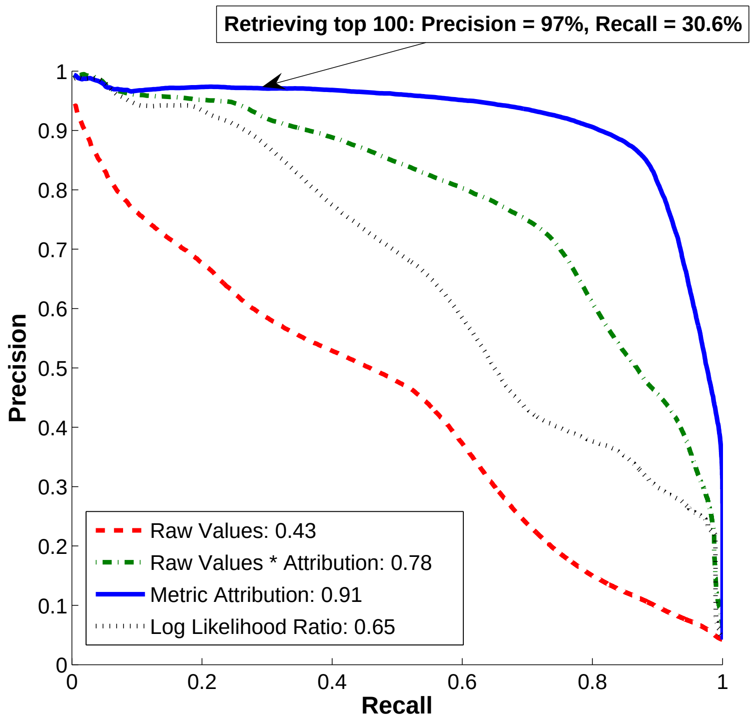


Figure 5.5: **Precision-recall graph for retrieval of the signatures of the IDC issue in the web-service production environment.** Methods based on metric attribution outperform the one relying on raw values significantly.

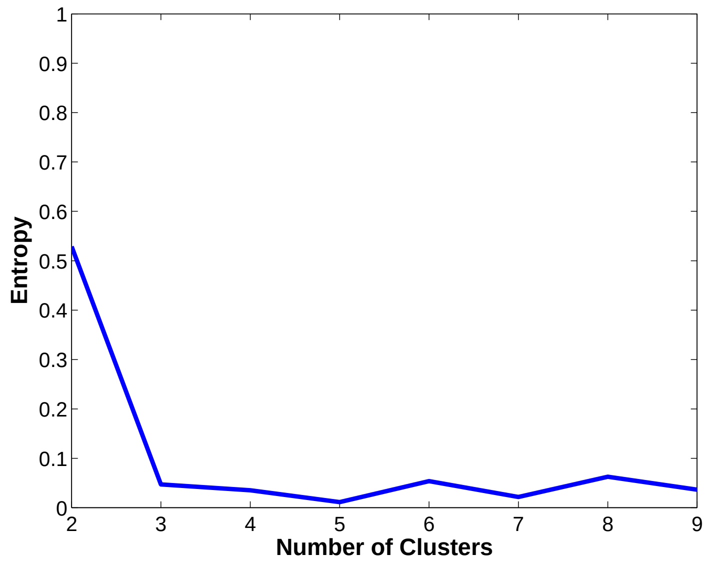


Figure 5.6: **Clustering on the labeled data from TESTBED.** As long as the number of clusters is greater than the number of different problem labels, the entropy is near zero. Zero entropy indicates pure clusterings.

only signatures with one type of label (and hence are of a better quality). Since the `TESTBED` trace is fully labeled (each signature of an epoch that violated SLO is labeled either `BUYSPREE`, `APPCPU`, or `DBCPU`), we can fully evaluate the effectiveness of our signatures for clustering. We vary the number of clusters (k) as it is a parameter that must be provided for the k-means clustering algorithm. Figure 5.6 shows that weighted average entropy of the clustering is close to 0 (ideal) as long as $k >= 3$, which is what we would intuitively expect from meaningful signatures and a good clustering algorithm. In addition, the stable entropy across different number of clusters is an indication of the robustness of the clustering output.

As explained in Section 3.4, we can still apply the notion of purity when labels are unavailable. However, instead of entropy measuring the purity of each cluster with respect to different labels, we base entropy calculations only on the percentage of violation and non-violation signatures in each cluster. Figure 5.7 shows the average weighted entropy of clustering `FT-TRACE` signatures over a range of cluster sizes. We

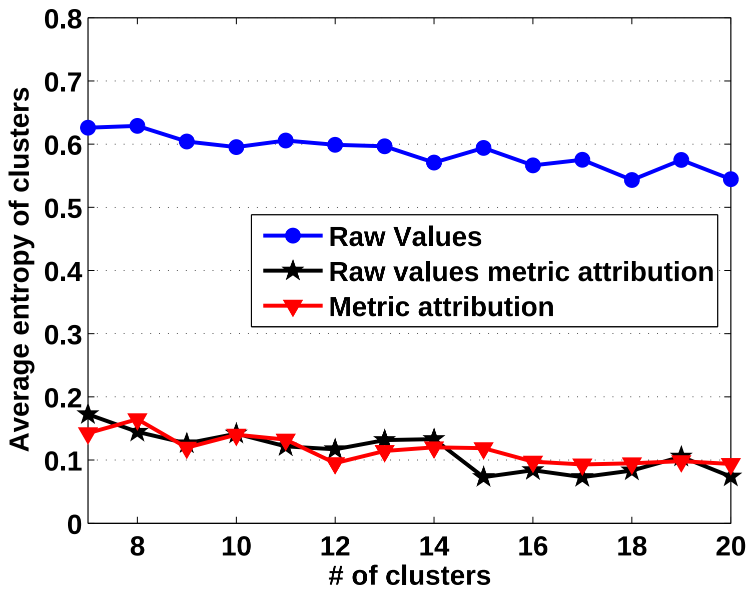


Figure 5.7: **Clustering on the data from FT-TRACE.** The signatures relying on information from metric attribution outperform those using only raw values. Clustering entropy is calculated without using labels since most violations were unlabeled.

see again the stability of entropy over k and conclude that the clustering is robust (as we increase the number of clusters, existing clusters are being subdivided rather than new ones being created).

We demonstrate that the clustering is meaningful for the case of nine clusters. Table 5.5 shows the number of elements belonging to each SLO state (compliance or violation) in each cluster. Note that for 7 of the clusters, comprising 90% of the 5 minute epochs in a trace collected over a month, the vast majority of elements in each cluster correspond to either compliance or violation. In addition these clusters are different from one another. Table 5.6 depicts the cluster centroids (with a subset of the metrics) for four of the clusters from Table 5.5: clusters 4 and 7, which contained only compliance signatures, and clusters 1 and 3, which contained mostly violation signatures. Note that the compliance centroids deem most metrics as inversely attributed (value of -1), while the violation centroids deemed some of the metrics as attributed (value 1), and most others as not attributable. We also see the difference between

Cluster #	# violations	# compliances	Entropy
1	552	27	0.27
2	225	0	0.00
3	265	2	0.06
4	0	1304	0.00
5	1	1557	0.00
6	0	1555	0.00
7	0	1302	0.00
8	216	274	0.99
9	100	128	0.99

Table 5.5: **Example of a clustering instance using metric attribution based signatures on FT-TRACE data ($k = 9$).** The first column is a count of number of violation instances, the second shows the number of compliance instances, and the third shows the cluster entropy based on the purity of the cluster. Cluster 3 contains almost all the instances corresponding to the IDC problem.

the centroids of the "violation" clusters (1 and 3) with respect to the metrics that are deemed attributed. Cluster 1 deemed the Database tier CPU utilization (DB1 cpubusy) as attributed but assigned a -1 value for the application server CPU utilization (gbl cpu total util). In contrast, the centroid of cluster 3 deemed the application server CPU as attributed, together with the number of alive processes and active processes on the application server. As discussed earlier, most members of cluster 3 were labeled as the IDC problem, which had the symptom of high application server CPU utilization and high number of alive and active processes. These differences point to the symptoms of the members in each cluster and define the syndrome of a group of signatures.

Given this clustering, recurrent problems can be identified by looking at the time of occurrence of the signatures in each cluster. Figure 5.8 depicts the instances of cluster 1, 2 and 3 overlaid in time on the performance indicator graph. Cluster 3 is recurrent, and as mentioned earlier, we verified with the IT operators that the periods defined by "Cluster 3" coincided with the manifestation of the IDC problem according to their records. Thus, had they had this tool, they could have easily identified the problem as a recurring one since its symptoms matched those of the signatures. In addition, the clustering discovered another undiagnosed recurring problem (Cluster 1), with

Metric	Cluster 1	Cluster 3	Cluster 4	Cluster 7
gbl app cpu total util	0	1	-1	-1
gbl app disk phys io	0	0	1	1
gbl app alive proc	0	1	0	-1
gbl app active proc	0	1	1	-1
gbl app run queue	0	0	-1	-1
gbl app net in packet rate	0	0	-1	1
gbl app net out packet rate	1	1	-1	-1
gbl app mem util	0	0	-1	-1
gbl app mem sys util	0	0	-1	-1
DB1 cpu util	1	1	-1	-1

Table 5.6: **Comparison of the centroid values for four clusters** (clusters 1, 3, 4 and 7 from Table 5.5), two containing mostly compliance signatures (clusters 4 and 7) and two containing mostly violation signatures (clusters 1 and 3). Cluster 3 contained mostly signatures of the IDC problem. Note the difference between cluster 1 and cluster 3 in terms of metrics that are attributed and those not attributed.

the symptom of higher Database CPU utilization (average of approximately 60% compared to approximately 20% in most other times), while at the same time all application server utilization metrics were not attributed, and were in fact normal. This problem remains undiagnosed to this date, and did not appear again in the following months, however, if it appears again, these past instances would be retrieved and perhaps help prioritize finding a solution or the root cause of the problem.

Note that neither the signature construction process, nor the clustering algorithm, considers temporal information. That is, there is no information contained within signatures that correspond to the time epochs they represent. Since we intuitively expect that problems close together in time are more likely to share a common root cause, the fact that these clusters tend to be grouped together in time is another strong indication that our signatures are meaningful.

5.5.3 Case study: Leveraging Signatures Across Installations

In this section we provide evidence that the signatures collected at various sites and systems can be leveraged during the diagnosis of performance problems. In particular, we show that diagnosis of a performance problem can be aided by querying for similar

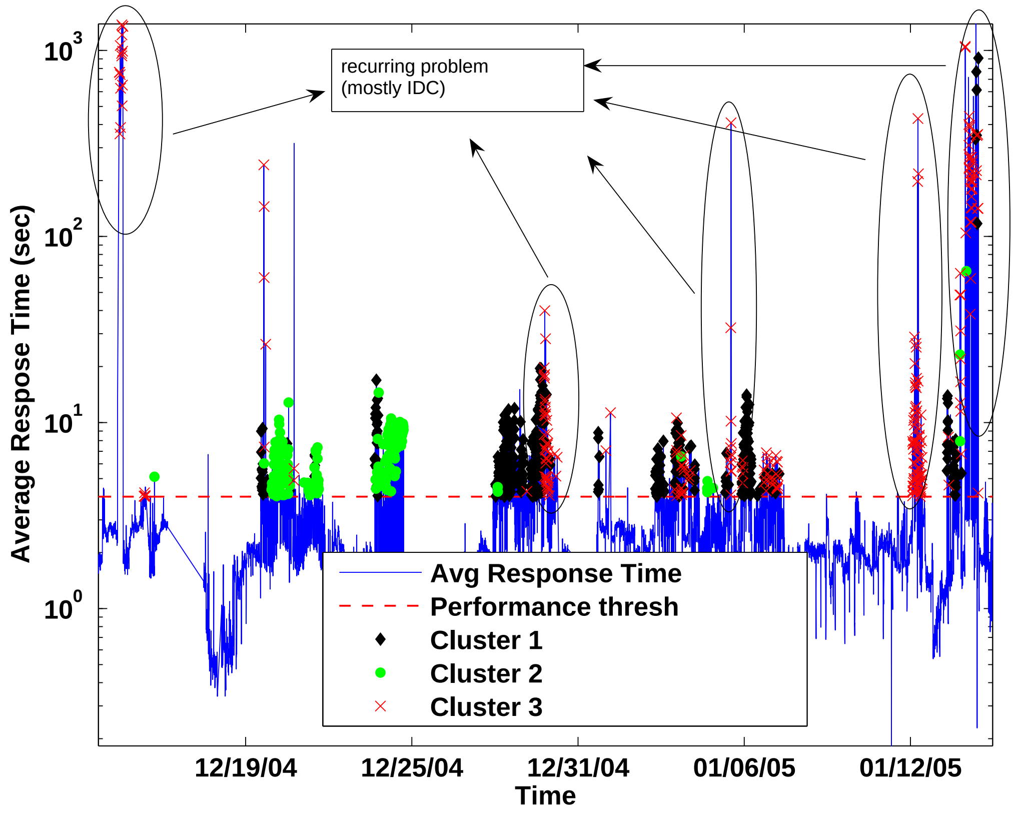


Figure 5.8: **Instances of the three "pure" abnormal clusters (clusters 1, 2 and 3 from Table 5.5), overlaid on average response time.** Cluster 3 comprises mostly of the instances labeled as the IDC problem. Cluster 1 is another recurrent problem with the symptom of high Database CPU load and low Application server utilization, while Cluster 2 has memory and disk utilization metrics on the Application server as attributed metrics to the performance problem.

(or dissimilar) signatures collected at different sites or machines.

In the process of diagnosing the `IDC` problem, which was observed on the Americas site, the debugging team investigated whether the same problem occurred in the Asia-Pacific region as well. In particular, they hypothesized that it did occur during a failover period on December 18th, 2004, in which the transactions from the Americas cluster were being sent to the Asia systems hosting the FT application. A high percentage of the transactions were violating the SLO on one of the Asia cluster machines during the first 100 minutes of the failover period. The debugging team suspected that the cause was the same `IDC` problem, and labeled it as such. Our signature database included signatures on traces collected on that day. We then performed the following query: are the signatures labeled as the `IDC` problem on the Americas site similar to the signatures collected during the failover period at the Asia site?

As Figure 5.9 shows, the result of the query was that the signatures of the Asia failover period are very different from the signatures of the `IDC` problem. Key metrics that were highly attributed in one were not attributed in the other. Of the metrics that were attributed in both, only transaction volume was similar in its attribution signal for the signatures from the two sites.

Upon close inspection of the attributed metrics from one of the Asia machines and the transaction mix on that machine, we quickly arrived at a different diagnostic conclusion for the Asia problem. Due to the failover from the Americas system, Asia1 was experiencing higher transaction volumes, and during the initial phase of the failover, it was experiencing higher response times (see Figure 5.10). During this initial phase, Asia1 was seeing a high transaction volume of one type of transaction (referred to as the XYZ transaction in Figure 5.11) that it normally does not see. The SQL statements associated with this unusual transaction type were not prepared or cached on the Asia machines, leading to more database overhead (Figure 5.12), higher response times, and ultimately SLO violations. This diagnosis was accepted by the diagnostics team; the repair consists of priming the database and middleware caches for the new transaction type before a planned failover. As a result of this experience, we were able to replace the false labels originally provided for that trace data with a

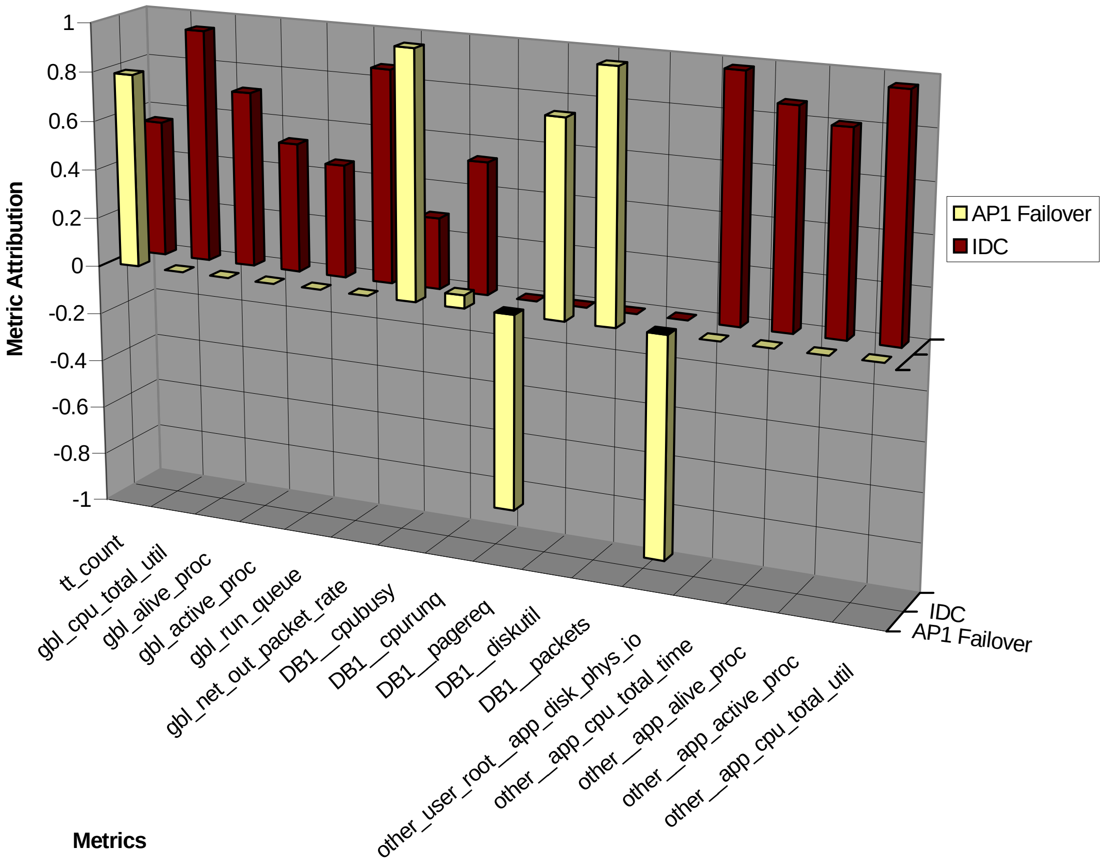


Figure 5.9: **Comparing the signatures from the Asia failover period and the IDC problem in the Americas.** The bars for each metric show the mean attribution value for the signatures in each period. For metrics where there are fewer than two bars shown, a missing bar means that the metric was not selected by any model that predicted the violation for this period. Metrics whose name does not begin with "DB1" are from the application server.

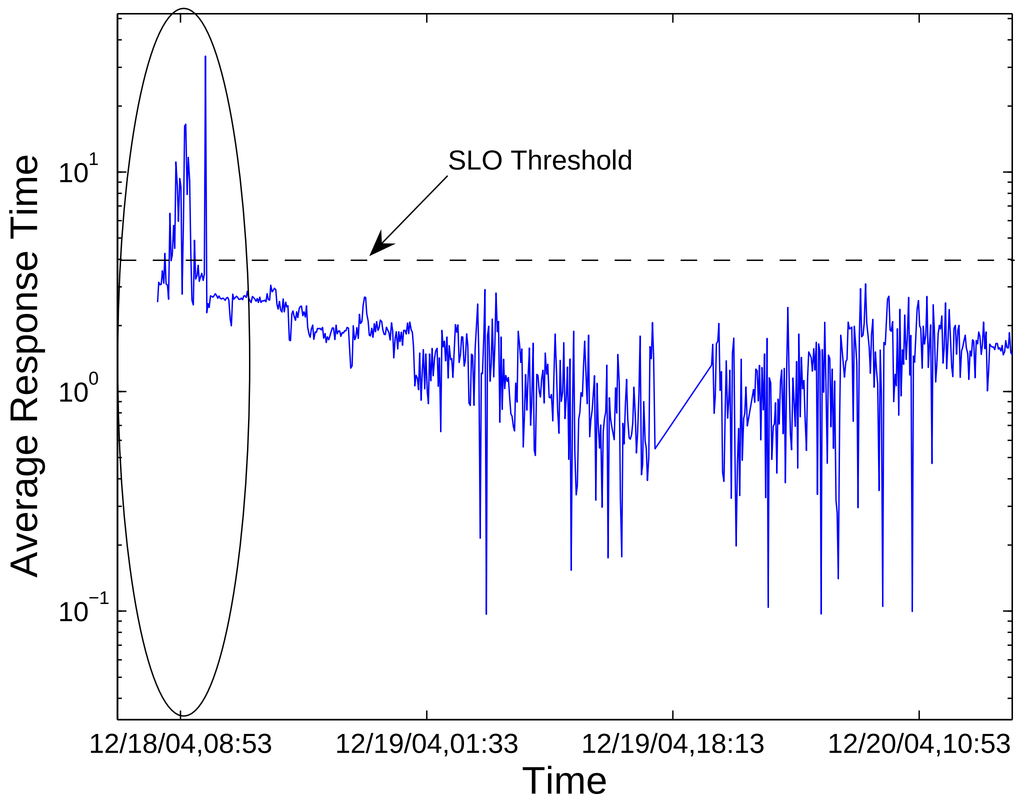


Figure 5.10: **Average response time during the Asia1 failover period.**

new and correct label explaining the problem and describing the required repair.

5.6 Performance

There are five points where our approach adds computation cost or other overhead that may impact performance of a system: overhead of collecting data, construction of models for metric attribution, signature computation, clustering, and retrieval. Our system data is collected by a commercial tool that is widely deployed in industry; the tool is designed to minimize performance impact on the observed system, and at any rate the widespread use of such tools represents a sunk cost. Updated system data is coalesced and reported periodically, generally in 1 or 5 minute epochs. The entire

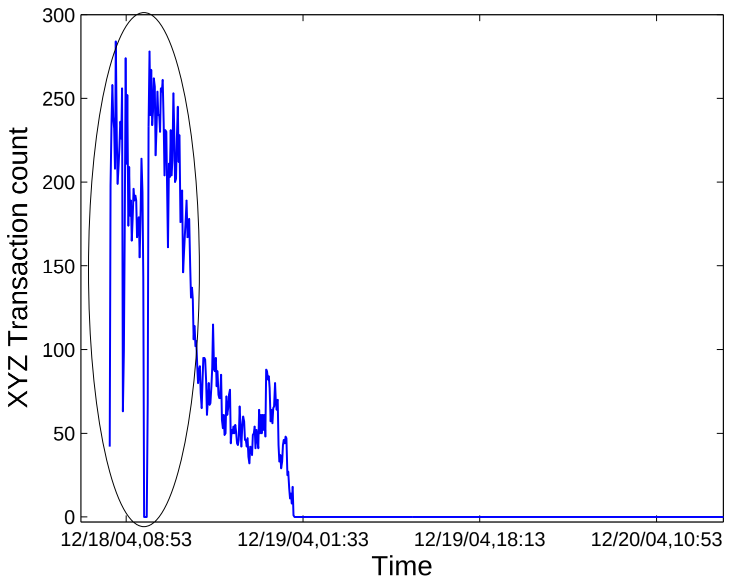


Figure 5.11: **Throughput for the XYZ Transaction during the Asia1 failover period.** XYZ transactions are usually never seen in Asia.

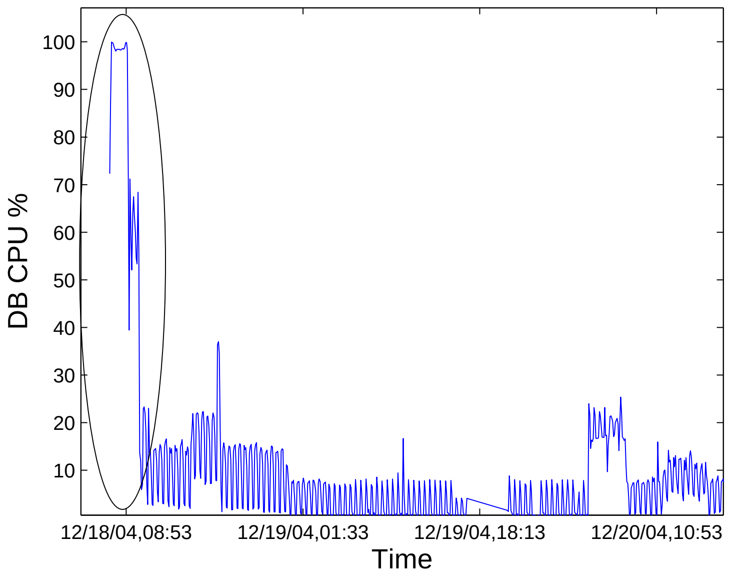


Figure 5.12: **CPU utilization on DB server during Asia1 failover period.** The utilization was unusually high at the beginning of the failover period. Once the caches were warmed, CPU utilization returned to 20% or lower.

	Testbed Trace	**FT Trace**
Epoch length	1 min	5 mins
Number of epochs	1079	7507
Constructing metric attribution based signatures	65 secs	315 secs
Retrieving top 100 matching signatures	< 1 sec	< 1 sec
k-means clustering ($k = 5$)	2 secs	9 secs

Table 5.7: **Performance impact of signature construction and use.** All timing measurements were taken on a Pentium 4 2.0 GHz laptop with 1 GB RAM. Algorithms were implemented using Matlab 7.0.

process of generating signatures for one month of data (7507 epochs of 5 minutes each), as well as analyzing the signatures using clustering and retrieval techniques, takes a matter of minutes on a Pentium 4 based laptop computer using Matlab prototype code. Detailed timings for each operation is giving in Table 5.7 for both `FT-TRACE` and `TESTBED` traces. We conclude that signature generation can proceed in soft real time, and analysis with clustering or retrieval is fast enough to be done at will.

5.7 Limitations of signature evaluation

For many traditional areas where statistical and machine learning techniques have been applied, determining the ground truth of data sets for evaluation purposes has been a matter of hiring human experts. The correct interpretation for handwritten letter samples is easily decided by humans. However, in the domain of system problem diagnosis, obtaining ground truth is extremely difficult. Even if operators are certain about the root cause of a failure, there is no exact notion of what the correct signature for that system state should be. We must rely on approximate human labeling to evaluate if signature similarity does indeed infer problem similarity.

It is a fact of life in the IT trenches that labels are scarce and imperfect. Part of the reason for this is the lack of advanced diagnostic tools, precisely the hole that we are attempting to fill. Additionally, administration of different subsystems or tiers of an application may be delegated to different individuals distributed across an organization, as we experienced when investigating the problem and resolution described in

Subsection 5.5.3. Our experiences with other companies running multi-tier applications confirm that there is often no single administrator responsible for understanding the end-to-end paths through an application. An unfortunate consequence is a frequent lack of clear agreement on what the true cause of a problem is or was: forensic data may be discarded before the realization that it is needed, and each operator is typically focused on either debugging or exonerating his or her piece of the system.

Fortunately, problem labels are only required for complete evaluation of our technique. Even lacking labels, partial evaluation is possible through the purity score of clustering violation and compliance signatures together. Therefore, the impreciseness of labeling and the limits that that imposes on signature evaluation should not be considered a limitation of our overall approach. Nevertheless, a main focal point of our technique is to better enable operators to leverage previous diagnostic efforts. Therefore, we hope that the availability of a systematic way to exploit labels will encourage a change in best practices that makes labels more prevalent and reliable. In Chapter 8 we discuss how operators can best leverage signatures for problem diagnosis.

5.8 Summary

This chapter described the systems and traces from those systems with which we used for evaluation of our signatures based approach.

- Traces were collected from two systems, an experimental testbed 3-tiered system running the Petstore sample application, and a globally-distributed enterprise system serving business-critical services to real customers. A widely used commercial software, HP OpenView was used for monitoring and reporting of metric data.

- The experimental testbed system was injected with 3 faults that tended to cause SLO violations. The BUYSPREE scenario involved increasing workload by adjusting the buy to browse ratio. The APPCPU and DBCPU scenarios consists of parasitic process consuming only CPU cycles only the application server and the database server, respectively.

- The production system trace contained many violations. However, only some of these violations were ever diagnosed by the system operators. In particular, we focus on a problem termed Insufficient Data Connection (`IDC`) which occurred several times over the month or so that this trace covered.

- Using precision-recall behavior, we compared the effectiveness of four different signature compositions (raw metric values, metric attribution only, attribution and raw values, and loglikelihood ratio) on both the `TESTBED` and `FT-TRACE` data sets. We found that not only was attribution based signatures superior in all cases, but that the accuracy of retrieving similar problem instances (97% for top 100 search for `IDC`) was more than adequate to be of use to a system operator.

- We verified the ability of the k-means clustering algorithm to group together different signatures for the `TESTBED` trace according to problem label, generating clusters with very low entropy (high purity). Since the `FT-TRACE` data was not fully labeled (only `IDC` problem labels were available), we could only partially evaluate signature clustering on that trace, utilizing entropy calculations based on the ratio of compliance signatures and violation signatures in each cluster. We found that entropy to be low and does not vary when increasing the number of clusters and thus we conclude that the clustering is robust.

- We successfully leveraged signatures across different sites of the FT service to correct an erroneous initial diagnosis of a performance problem after a failover situation.

- Generating metric attribution based signatures for over a month of data (from service FT) took only minutes. Clustering and retrieval of signatures takes on the order of seconds. Therefore, we conclude that our approach can be used in real time as a system is running.

- Unlike some other domains, there is often no ground truth for what signatures should capture. We must use operator provided labels (annotations of diagnosed root cause) to evaluate our techniques. These labels are often incomplete and imprecise due to the difficulties in diagnosing complex systems today. However, this impacts only the accuracy of evaluation and rather than the effectiveness of our approach. When labels are lacking, clustering purity, as well how closely grouped in time clusters are, can be used as indirect measures of signature quality.

Chapter 6

Adapting to change

In order to construct signatures with the good retrieval and clustering properties evidenced in the last chapter, several major modifications were needed to the original metric attribution procedure described in Section 4.3. These changes will be described in the next few chapters. This chapter focuses on improvements allowing the statistical models from which attribution is computed to adapt to changing system behavior. The majority of this chapter has been previous published in [53] and was joint work with Ira Cohen, Moises Goldszmidt, and Julie Symons of Hewlett-Packard Labs.

6.1 Dynamic system behavior

Complex systems today are highly dynamic. Frequent software and hardware changes not only make system problems more likely, but they can also fundamentally alter the way a system behaves. Patterns of behaviors captured by statistical models (or manually noticed by system operators) could change significantly over a period of days or even hours. The experiments in [15] hinted at the need for metric attribution to adapt, as under different conditions, different metrics, or different thresholds, the metrics deemed attributable varied significantly. This is unsurprising since a system behavior may be altered by not only changes in the infrastructure, but also changes in the workload due to surges or rebalancing (due to transient node failures), faults in the hardware, and bugs in the software, among other factors.

A single Bayesian network classifier is designed to capture one particular correlation pattern between high level system behavior (i.e. SLO compliance state) and lower level metrics. Simple experiments showed that an accurate classifier induced on data captured while a system was under one workload pattern were very inaccurate (poor balanced accuracy, see Section 4.2.1) when evaluated on data from the same system but under a different workload condition. The validity of metrics selected as attributable by a statistical model depends heavily on the ability of that model to accurately capture the underlying behavioral patterns in that data. Clearly, we cannot simply induce a model from a period of system information and expect it to faithfully describe all possible behaviors of that system.

6.2 Approach: Ensembles of Models

This section presents a technique for maintaining an *ensemble* of models to enable adaptation of metric attribution computations to changes in system behavior. As a system runs we periodically induce new models and augment the ensemble if the new models capture a behavior that no existing model captures. The problem of adapting metric attribution is thereby reduced to the problem of managing this ensemble. Several management issues must be addressed. When should new models be induced? Which model(s) are the most relevant to current system behavior? How can we combine information from multiple models to provide a single attribution value for each metric?

We sketch our approach as follows. We treat the regularly reported system metrics as a sequence of data vectors. As new data are received, a sliding window is updated to include the data. Periodically, we score the balanced accuracy (BA) of existing models in the ensemble against the data in this sliding window and compare this to the BA of a new model induced using only the data in this window. Based on this comparison, the new model is either added to the ensemble or discarded.

For every epoch, we use the Brier score [7, 14] to select the most relevant model from the ensemble to use for metric attribution. The Brier score is related to the mean-squared-error often used in statistical fitting as a measure of model goodness. Equation 4.1 from Section 4.3 is still used to derive the attribution values. The rest of this section describes the rationale for this approach and the details of new model induction and ensemble usage.

6.2.1 Ensemble vs. Single Model

The observation that changes in workload and other conditions leads to changes in metric attribution relationships has been observed in real Internet services [27] and in our own experiments. In the context of probabilistic models in general and Bayesian network models in particular, there are various ways to handle such changes.

One way is to learn a single monolithic Bayesian classifier model that attempts to capture all variations in workload and other conditions. There are two problems with this approach. First, as new data about a system is collected, the monolithic model must be rebuilt including the new data. This means that the amount of data that has to be processed when inducing the model is ever increasing. However, due to the efficiencies of our techniques of model induction, the computational cost of this method may not be prohibitive. The second and more important issue with using a single monolithic model is that it simply does not capture multiple behavior patterns well. For example, a problem may initially increase I/O request rates to be much higher than normal. A different problem may manifest later that results in I/O requests being lowering than normal. If we attempted to model both these conditions in a single model, it would capture the fact that during SLO violating behavior, the I/O request rate may be high or low and therefore is not strongly correlated to SLO state. In actuality, that metric should be attributed for both of the problem and that would be the case if we used a different model for each condition. We used data collected on a system with two different workloads (these workloads are described in Section 6.3.1), one that mimics increasing but "well-behaved" activity (e.g. when a site transitions from a low-traffic to a high-traffic service period) and the second mimics unexpected surges in activity (e.g. a breaking news story). We will show that while a single classifier trained on both workloads had a balanced accuracy of 72%, utilizing two separate models, one trained on each workload, we achieve a BA of 88%.

A second approach, trying to circumvent the shortcomings of the single monolithic model, is to use a single model that is constantly updated with the most recent system data. The most significant issue with such an approach is in the fact that the single model does not have "long term" memory, that is, the model only captures short term system behavior, forcing it to re-learn conditions that might have occurred previously.

In the third approach, advocated by this dissertation, adaptation is addressed by using a scheduled sequence of model inductions, keeping an ensemble of models instead of a single one. The models in the ensemble serve as a memory of the system in case problems reappear. In essence, each model in the ensemble is a summary of the data used in building it. We can afford such an approach since the cost, in terms of computation and memory, of learning and managing the ensemble of models is relatively negligible: new system data typically arrives every one to five minutes in commercial products (e.g., HP OpenView), while learning a model takes on the order of several seconds, evaluating a model in the ensemble takes around 1 msec and storing a model involves keeping on the order of tens of floating point numbers[1]. The challenges of managing an ensemble lie in deciding when to add new models to the ensemble and how to combine the information from the ensemble's models.

6.2.2 Inducing and incorporating new models

The general procedure for inducing a statistical model remains the same as described in Section 4.3. It is important to note that the feature selection process is repeated for every new model induced. Recall that this process is mainly needed to deal with the dimensionality problem, where the number of data samples needed to induce robust models increases exponentially with the dimension (which is directly related to the number of metrics under consideration) of the problem. Repeating this for each model is done to avoid the situation where metrics deemed relevant at a previous time are never again considered even though they may become relevant in the future. Feature selection has been improved by incorporating a beam search algorithm rather than

[1]These performance figured were generated on a Pentium 4 2.0 GHz laptop with 1GB of ram using Matlab 7.0 prototype code.

simple greedy search, which provides some robustness against local minima [29].

Models are induced periodically over a data window D consisting of vectors of n metrics at some time t, $\vec{M}_t = [m_1, \ldots, m_n]$ and the corresponding SLO state (s^- or s^+). Once the model is induced, we estimate its balanced accuracy (Eq. 4.1) on the data set D using tenfold crossvalidation [28]. We also compute a confidence interval around the new BA score. If the new model's BA is statistically significantly better than that of all models in the existing ensemble, the new model is added to the ensemble; otherwise it is discarded. In our experiments, models are never removed from the ensemble. Although any caching discipline (e.g. Least Recently Used) could be used to limit the size of the ensemble, we did not study this issue in depth because evaluating models takes milliseconds and their compact size allow us to store thousands of them[2], making the choice among caching policies almost inconsequential.

Algorithm 2 Learning and Managing an Ensemble of Models

```
Input: WindowSize and Frequency of Induction
Initialize Ensemble to {$\phi$} and CurrentWindow to {$\phi$}
for every new sample do
  update CurrentWindow to include new sample (and discard oldest sample if
  necessary)
  if CurrentWindow has enough samples and it is time to induce a new model
  then
    Do feature selection and induce new model $M$ on CurrentWindow
    Compute accuracy of $M$ using crossvalidation.
    For every model in Ensemble compute accuracy on CurrentWindow.
    if accuracy of $M$ is statistically significantly higher than the accuracy of all
    models in the Ensemble then
      add new model to Ensemble.
    end if
  end if
  Compute Brier score (Eq. 6.1) over CurrentWindow for all models in Ensemble
  Perform metric attribution using model with best Brier score
end for
```

[2]Also note that the more models in an ensemble, the less likely that a new model will be added to it since it must be significantly more accurate than all current models.

Algorithm 2 describes in the detail the algorithm for managing the ensemble of models. There are two main parameters to this process that need to be set. First and foremost, the size of the data window D for use in inducing new models must be considered. Choosing too large a window may increase the number of patterns that a single model attempts to capture, resulting in less accurate models (exactly the problem with using a single monolithic model). However, too small a window may result in overfitting of the data and thus a non-robust model. Lacking closed-form formulas that can answer these questions for Bayesian classifiers given potentially complex system behaviors, we follow the typical practice in machine learning of using an empirical approach. In Section 6.3, we use *learning surfaces* to characterize minimal data requirements for the models in terms of number of data samples required as well as proportions of SLO violation versus compliance epochs in a data window. The second, less important parameter is the frequency of inducing new models. Settings of this parameter make a tradeoff between unnecessary computation and potentially an adaptation rate that is too slow for rapidly changing system behaviors.

6.2.3 Utilizing multiple models

When using a single model, accuracy is determined by the accuracy of that model in correctly predicting SLO state from metric data. In addition, attribution is determined by querying that model to determine the log-likelihood ratios of each metric's value during some epoch. To compute the accuracy and attribution when using an ensemble approach, we follow a *winner takes all* strategy: we select the "best" model from the ensemble, and then proceed as in the single-model case.

Although the machine learning literature describes methods for weighted combination of models in an ensemble to get a single prediction, our situation differs from those cases because our models are each trained on different windows of data and are designed to capture different types of behaviors that a system can exhibit. Therefore, our goal is not to merge the opinion of many models, but rather select the most appropriate model at a given point in time. We accomplish this task by utilizing the Brier score[7].

For each model in the ensemble, we can compute its Brier score over a short window of past data, $D = \{d_{t-w}, \ldots, d_{t-1}\}$, where t is the present sample, making sure D includes samples of both SLO compliance and violation. The Brier score is the mean squared error between a model's probability of the SLO state given the current metric values and the actual SLO state, i.e., for every model in the ensemble, Mo_j:

$$BS_{Mo_j}(D) = \sum_{k=t-1}^{t-w} [P(s^+|\vec{M} = \vec{m}_k; Mo_j) - I(s_k = s^+)]^2, \tag{6.1}$$

where $P(s^+|\vec{M} = \vec{m}_i; Mo_j)$ is the probability of the SLO state being in violation of model j given the vector of metric measurements and $I(s_k = s^+)$ is an indicator function, equal to 1 if the SLO state is s^+ and 0 otherwise, at time k. For a model to receive a good Brier score (best score being 0), the model must be both correct and confident (in terms of the probability assessment of the SLO state) in its predictions.

Although in classification literature the accuracy of the models is often used to verify the suitability of an induced model [5, 10, 17], we require a score that can select a model based on data that we are not sure has the same characteristics as the data used to induce the models. Since the Brier score uses the probability of prediction rather than just {0,1} for matching the prediction, it provides us with a finer grain evaluation of the prediction error. Our experiments consistently show that using the Brier score yields better accuracy, confirming the intuition expressed above.

Note that we are essentially using a set of models to capture the drift in the relationship between the low level metrics and high level behavior as defined by the SLO state. The Brier score is used as a proxy for modeling the change in the probability distributions governing these relationships by selecting the model with the minimal mean squared error over the current data window.

6.3 Evaluation

We validated this ensemble technique using the experimental testbed system described in Section 5.2. In addition to the three previously described workloads `BUYSPREE`, `APPCPU`, and `DBCPU`, we introduce two new workloads, `RAMP` and `BURST`, in the next section. We then report the results comparing the ensemble technique to: (a) a single monolithic model trained using all experimental data, and (b) an "oracle" that knows exactly when and how the workload varies among the five workload types. The oracle method induces workload-specific models for each type and invokes the appropriate model during testing; while clearly unrealistic in a real system, this method provides a qualitative indicator of the best we could do. Last, we present the result of using learning surfaces to characterize the amount of data needed to induce robust classifiers.

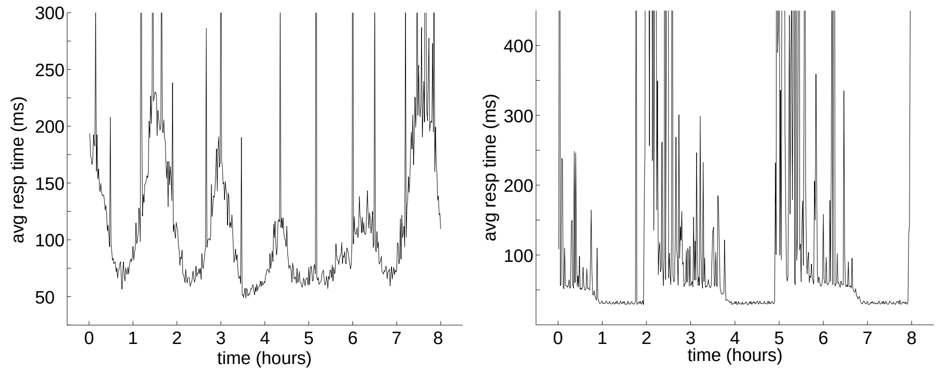
6.3.1 Workloads

For the `RAMP` workload, the number of concurrent client sessions was gradually ramped up. An emulated client was added every 20 minutes up to a limit of 100 total sessions. Individual client request streams were constructed so that the aggregate request stream resembles a sinusoid overlaid upon a ramp. The average response time of the web service in response to this workload is depicted in Figure 6.1. Each client session follows a simple pattern: go to main page, sign in, browse products, add some products to shopping cart, check out, repeat.

The `BURST` workload has two components. The first httperf creates a steady background traffic of 1000 requests per minute generated by 20 clients. This second is an on/off workload consisting of hour-long bursts with one hour between bursts. Successive bursts involve 5, 10, 15, etc client sessions, each generating 50 requests per minute. The intent of this workload is to mimic sudden, sustained bursts of increasingly intense workload against a backdrop of moderate activity. Each step in the workload produces a different plateau of workload level, as well as transients during the beginning and end of each step as the system adapts to the change.

6.3.2 Results

We learn the ensemble of models as described in Algorithm 2 using a subset of the available data. The subset of data are the first 4-6 hours of each workload. We keep the last portion of each workload as a test (validation) set. Overall, the training set



(a) (b)

Figure 6.1: **Relevant sequences of average web server response time over 1-minute windows when the test system was subjected to the (a) RAMP workload (b) BURST workload**

is 28 hours long (1680 epochs, each epoch being 1 minute) and the test set is 10 hours long (600 epochs). The training set starts with alternating two hour sequences of `DBCPU`, `APPCPU`, and `BUYSPREE`, with `DBCPU` and `APPCPU` represented three times and `BUYSPREE` twice. This is followed by six hour sequences of data from `RAMP` and `BURST`. The result is a training set which has five different workload conditions some of which are repeated several times. Testing on the test set with the ensemble amounts to the same steps as in Algorithm 2, except for the fact that the ensemble is not initially empty (but has all the models trained on the training data) and no new models are added at any point. The accuracies provided by this testing procedure show how generalizable the models are on unseen data with similar types of workloads.

Table 6.1 shows the balanced accuracy, false alarm and detection rates, measured on the validation data, of our approach compared to the single monolithic model and the workload specific models. We present results for the ensemble of models with three different training window sizes (80, 120, and 240 samples). We see that for all cases: (a) the ensemble's performance with any window size is significantly better than the single-model, (b) the ensemble's performance is robust to wide variations of the training window size, and (c) the ensemble's performance is slightly better, mainly in terms of detection rates, compared to the set of five individual models trained on each of the workload conditions we induced.

	metrics chosen	avg attr metrics	BA	FA	Det
Ensemble W80	64	2.3	95.67	4.19	95.53
Ensemble W120	52	2.5	95.12	4.84	95.19
Ensemble W240	33	3.7	94.68	5.48	94.85
Workload specific models	9	2	93.62	4.51	91.75
Single model	4	2	86.10	21.61	93.81

Table 6.1: **Summary of adaptation performance results.** First three rows show results for ensemble of models with different size training window (80, 120, 240 samples). Metrics chosen represents all of the metrics used in all models in an ensemble. Average attributed metrics refers to the average number of metrics found to be attributable for epochs of SLO violation.

The last observation (c) is at first glance puzzling, as intuition suggests that the set of models trained specifically for each workload should perform best. However, some of the workloads on the system were quite complex, e.g., `BURST` has a ramp up of number of concurrent sessions over time and other varied conditions. This complexity will be further characterized by learning surfaces, where we will see that it takes many more samples to capture patterns of the `BURST` workload, and with lower accuracy compared to the other workloads. The ensemble of models, which is allowed to learn multiple models for *each workload*, is better able to characterize this complexity than the single model trained with the entire dataset of that workload. Intuitively, this also makes sense since a single workload "pattern" applied to a system does not imply that it will exhibit a single behavior pattern in response to that workload. For the `BURST` workload, it seems likely that system behavior will be quite different depending on the intensity of a burst.

The table also shows the total number of metrics included in the models and the average number of metrics attributable to specific instances of SLO violations. The ensemble chooses many more metrics overall because models are trained independently of each other, which suggests that there is some redundancy among the different models in the ensemble. However, only a handful of metrics are attributed to each instance of an SLO violation, therefore attribution is not affected by the redundancy.

To show how the ensemble of models adapts to changing workloads, we store the

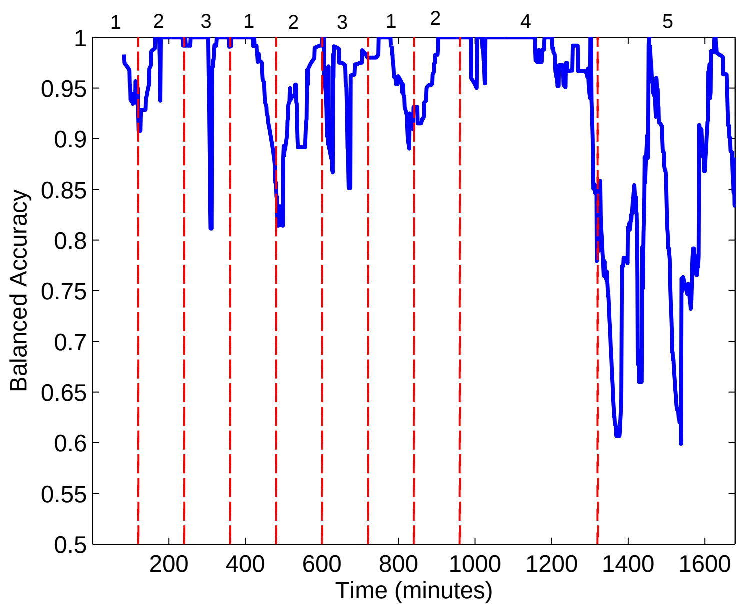


Figure 6.2: **Balanced accuracy of ensemble of models during training.** Vertical dashed lines show boundaries between workload changes. Numbers above figure enumerate which of the five types of workload corresponds to each period (1=DBCPU, 2=APPCPU, 3=BUYSPREE, 4=RAMP, 5=BURST).

accuracy of the ensemble of models as the ensemble is trained. The changes in the ensemble's accuracy as a function of the number of samples is shown in Figure 6.2. There are initially no models in the ensemble until enough samples of violations are observed. The ensemble's accuracy remains high until new workloads elicit behaviors different from those already seen, but adaptation is quick once enough samples of the new workload have been seen. An interesting situation occurs during adaptation for the last workload condition (`BURST`, marked as 5 in the figure). We see that the accuracy decreases significantly when this workload condition first appears, but improves after about 100 samples. It then decreases again, illustrating the complexity of this workload and the need for multiple models to capture its behavior, and finally recovers as more samples appear. It is worth noting that there is a tradeoff between adaptation speed and robustness of the models, e.g., with small training window, adaptation would be fast, however, the models might not be robust to overfitting and will not generalize to new data.

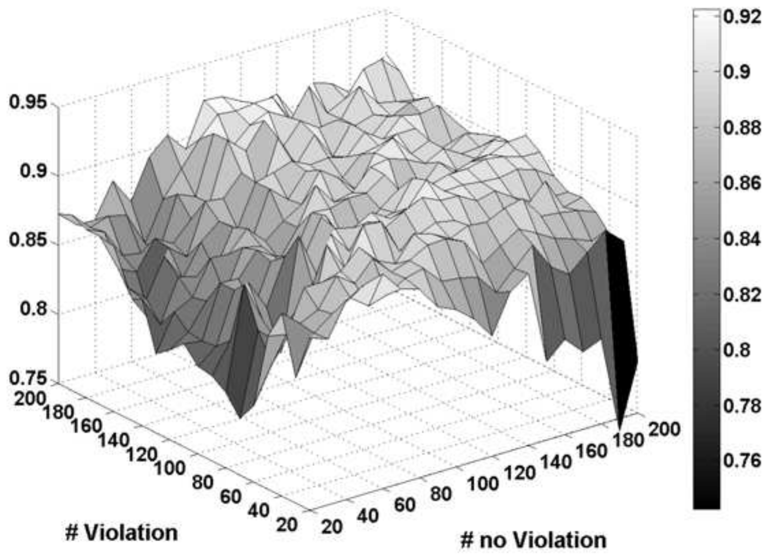


Figure 6.3: **Learning surface for RAMP experiment showing balanced accuracy.** The color map on each figure shows the mapping between color and accuracy. Each quad in the surface is the balanced accuracy of the right bottom left corner of the quad.

To test how much data is needed to learn accurate models, we take the common approach of testing it empirically. Typically, in most machine learning research, the size of the training set is incrementally increased and accuracy is measured on a fixed test set. The result of such experiments are learning curves, which typically show what is the minimum training data needed to achieve models that perform close to the maximum accuracy possible. In the learning curve approach, the ratio between violation and non-violation samples is kept fixed as the number of training samples is increased. Additionally, the test set is usually chosen such that the ratio between the two classes is equivalent to the training data. These two choices can lead to an optimistic estimate of the number of samples needed in training, because real applications (including ours) often exhibit a mismatch between training and testing distribution and because it is difficult to keep the ratio between the classes fixed.

To obtain a more complete view of the amount of data needed, we use Forman and Cohen's approach [20] and vary the number of violations and non-violations in

Workload	# of violation epochs	# of non-violation epoch	max BA(%)
RAMP	90	80	92.35
BURST	180	60	81.85
BUYSPREE	40	40	95.74
APPCPU	20	30	97.64
DBCPU	20	20	93.90

Table 6.2: **Minimum sample sizes needed to achieve accuracy that is at least 95% of the maximum accuracy achieved for each workload condition.**

the training set independently of each other, testing on a fixed test set that has a fixed ratio between violations and no violations. The result of this testing procedure is a *learning surface* that shows the balanced accuracy as a function of the ratio of violations to non-violations in the training set. Figure 6.3 shows the learning surface for the `RAMP` workload we described in the previous section. Each point in the learning surface is the average of five different trials. Examining the learning surface reveals that after about 80 samples of each class, we reach a plateau in the surface, indicating that increasing the number of samples does not necessarily provide much higher accuracies. The surface also shows us that small numbers of violations paired with high numbers of non-violations results in poor BA, even though the total number of samples is high; e.g., accuracy with 200 non-violations and 20 violations is only 75%, in contrast to other combinations on the surface with the same total number of samples (220) but significantly higher accuracy.

Table 6.2 summarizes the minimum number of samples needed from each class to achieve 95% of the maximum accuracy. We see that the simpler `APPCPU` and `DBCPU` workloads require fewer samples of each class to reach this accuracy threshold compared to the more complex `RAMP` or `BURST` workloads.

Even though we based the evaluation of our ensemble method on the accuracy of predicting SLO state from metric values, inferring the SLO state is clearly not the goal of metric attribution (since SLO state can simply be measured directly). Rather, the end goal is allowing attribution values to be used as the basis for signatures that can aid operators in diagnosing problems. Although we cannot prove that improving the accuracy of models used for metric attribution directly improves the quality of signatures, models that do not represent system behavior well (i.e. low prediction accuracy) clearly can not be counted on for useful information for signatures. We rely on the evaluation in Chapter 5 to conclude that signatures based on metric attribution generated using this ensemble method do indeed offer helpful information for system troubleshooters.

6.4 Summary

This chapter explored the rationale, method, and performance of employing an ensemble of models to allow metric attribution to adapt to changes in system behavior.

- Metric attribution calculations rely on inducing models of system behavior. When system behavior inevitably changes (due to workload changes, infrastructure changes, or software bugs, among many other factors), the model must change as well for metric attribution results to be meaningful.

- Using a single monolithic model cannot adequately capture multiple system behaviors, while maintaining an adaptive model that only captures the most recent system behavior results in forgetting of past patterns that may repeat in the future. Our experiments demonstrated both of these cases.

- Our approach utilizes an ensemble of models. New models are periodically induced on a sliding window of training data as new system information is gathered. If the new models capture system behavior better than any current models in the ensemble, the new model is added to the ensemble. For each epoch, the model that represents recent system behavior most faithfully (measured using the Brier Score) is used for metric attribution.

- Using traces from a testbed system, we found the ensemble method performs significantly better than a single monolithic model approach. Surprisingly, it is also better than using workload specific models, as complex workloads may result in many different patterns of system behavior and the ensemble method can better capture those multiple patterns.

- Although ensemble performance is robust to the size of the training window, the accuracy and robustness of models produced do rely on the training window having enough data samples. We demonstrated how learning surfaces can be used to gauge how much data is enough. In addition, we showed that more complex workloads and conditions require greater number of samples to induce a good statistical model.

Chapter 7

Improving signature robustness

Signatures based on metric attribution capture the set of symptoms that define an undesirable system state (SLO violation). As such, signatures depend on not only the problem behavior itself, but also on the good behavior that is used for comparison. Therefore, the same problem may be characterized by different signatures if compared to different good behaviors. Unfortunately, this is detrimental to the utility of our approach, which is based on the ability to accurately infer similarity between problem states based on the similarity between the signatures. In this chapter, we explore this issue further and present a technique for mitigating its effects.

7.1 Problem: Good behavior not unique

Properly designed systems often must perform well under a wide variety of workload conditions. Consequently, metric values often exhibit great variability during normal operation. For example, suppose a service handles many different types of client transactions, some of which are CPU intensive while others are I/O intensive. While this service may be able to support many different transaction mixtures without violating its SLO, the value of metrics such as CPU Utilization would vary significantly depending on the exact workload. Furthermore, component failures or software bugs may result in a system operating with less efficiency, but still meeting its SLO, especially during periods of low overall utilization. Since metric attribution relies on the binary SLO state, such types of behavior would also be considered desirable.

Although the concept of signatures in general is independent of any particular method of generating them, we have shown in previous chapters that signatures based on metric attribution exhibit the best clustering and retrieval properties. That is, metric attribution signatures were the most accurate for inferring problem similarity from signature similarity. These results would definitely benefit from being able to keep signatures robust to variances in good behavior. The signature of a problem behavior compared to one particular good behavior may be quite different if compared to another good behavior. This implies that a failure with the same root cause occurring at two different times may produce significantly different signatures, which is clearly detrimental to clustering and retrieval performance.

But why can we not simply model all non-violations as a single monolithic good behavior? The answer is for the same reasons that we do not use a monolithic model to capture multiple problem behaviors. Our classifiers do not treat compliant behavior any different from problem behavior in terms of modeling. We established in the previous chapter that inducing a single model on multiple problem behaviors resulted in poor classification accuracy (see Section 6.2.1). Furthermore, a dynamic system changes over time and as more data is collected, even a monolithic version of good behavior may have to be adapted.

Our challenge will be to determine which good behaviors to use when generating signatures of problem behaviors with the goal of maximizing the likelihood that problems with similar root causes will exhibit similar attributions.

7.2 Naive approach

The naive approach to addressing this problem is to simply choose *any single* good behavior to compare all undesirable behaviors against. We refer to this reference good behavior as a *baseline*. However, the issue with this approach is that similarity inferences may not be consistent when using different possible baselines.

The variability in metric attribution due to using different good behaviors is not in and of itself problematic. Clustering and retrieval, the main signature usage methods, depend not on the absolute value of signatures, but rather the distance between signatures. That is, similarity is inferred using the distance between signatures of different problems. Therefore, it is reasonable to ask whether the technique of basing all signatures against the same baseline behavior could be effective. Unfortunately, distances between signatures are also dependent on the particular baseline selected.

The following is an example that illustrates this issue. Suppose there are two periods of SLO violations due to two different root causes. Assume that during the first period, CPU utilization is usually between 40% and 50% while during the second period, it is between 75% and 90%. The issue is that depending on exactly *which* baseline is used, this metric may be found attributable for both, neither, or only one of the problem periods. If CPU utilization was between 10% and 15% during the baseline, this metric would be considered attributed for both problems. On the other hand, had the baseline behavior exhibited utilization between 35% and 45%, the metric would be attributable for the second period but not the first. While a signature would consider every low level metric, this example just focuses on CPU utilization for simplicity, and the highlighted issue extrapolates to considering multiple metrics. Note that in this example, we would want signatures of the two periods to differ as they represent different root causes. However, this may or may not turn out to be case depending on the baseline behavior used. Consequently, the implication is that some baselines would result in more accurate signatures than others.

Figure 7.1 shows a histogram of different retrieval accuracies (as measured by the area under the precision-recall curve for the `IDC` problem in `FT-TRACE`) resulting from the use of different randomly selected baselines. By random, we mean that the baselines are selected by randomly choosing a starting epoch using a uniform distribution of all possible starting epochs in the dataset. The ending epoch of the baseline is determined by the starting epoch and the baseline length. We show baselines of 100 epochs and of 300 epochs. Although there is a small improvement in the average accuracy when using a longer baseline, there is great amount of variance in both cases. This pattern holds for even longer or shorter baselines. We conclude from these figures that we would not be guaranteed good retrieval accuracy by randomly choosing a single baseline.

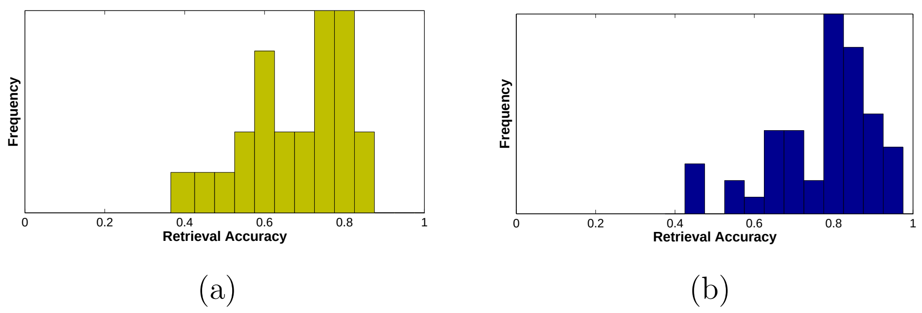


Figure 7.1: **Histograms of retrieval accuracies when using different baselines.** These histograms were produced by randomly selecting 50 different baselines of lengths **(a)** 100 epochs **(b)** 300 epochs. The Y-axis represents the frequency of achieving the retrieval accuracy specified by the X-axis, where retrieval accuracy represents the area under the precision recall curve for the IDC problem in FT-TRACE. Notice that good retrieval accuracy cannot be guaranteed by randomly choosing a baseline.

Could we possibly examine all possible baselines and choose the one that yields signatures with the highest accuracy? The answer in general is no, because reliable labels are required for evaluation. In real systems, labels are usually incomplete and only partially reliable if available at all. How, then, can one make an educated choice of which baseline to use in such a scenario?

7.3 Better approach: use baseline groups

Although we cannot guarantee good retrieval accuracy without reliable labels, we present an approach that allows for the weaker guarantee that attribution differences (which clustering and retrieval depends on) will not materially depend on the choice of baselines, provided the operators follows our guidelines in choosing them.

To help understanding of this challenge and our approach, we can draw an analogy to doctors diagnosing illnesses. The SLO state would corresponding to whether or not a patient is feeling sick. The low level metrics would include a patients vital stats (temperature, blood pressure, cholesterol levels, heart rate, etc). Computing a signature represents the first and most important aspect of diagnosis, defining the symptoms of the problem. For example, if a patient's temperature were at 102°

Fahrenheit, one of the symptoms of her illness would be a fever. Note that we can only arrive at the fever symptom by assuming that normal body temperature should be significantly less than 102° F, which is generally true for humans. However, normality for many other metrics (e.g. heart rate, blood pressure) is heavily dependent on factors such as a patient's age, race, sex, and lifestyle. Consequently, illness symptoms (signatures) depend not only on a patients' stats when they are sick, but also on the model of normality we choose as the basis for comparison. Doctors tend to rely on common medical knowledge and past experience to determine a sensible idea of normality. However, since design, composition, and workload can vary so much from system to system, we take the approach of deriving normality dynamically for each system.

We can further extend this analogy to show why certain baselines may result in more accurate signatures compared to using other baselines. For example, a patient's vitals taken immediately after running a marathon might be a poor choice for a baseline since her blood glucose level may be so low that if we accepted this condition as normality, almost all illness would have elevated blood glucose level as a symptom. Vitals taken after a large meal may also pose similar issues. Furthermore, it is not possible to determine which model of normality results in the most accurate signatures (i.e. where similar symptoms implies similar illness) without knowing exactly which illnesses are supposed to be the same (i.e. labeled data).

The technique we advocate to address this challenge is simply to use *multiple* baselines. Recall the ensemble learning procedure described in the previous chapter. As new data is collected, models are constructed on a moving window to include this new data. These models capture the differences between the SLO violations and SLO compliances in that window. We alter this process such that only the SLO violations in these moving windows are used. Instead of comparing that behavior to the compliance behavior within that same window, we compare it to multiple pre-selected good behaviors. These good behaviors will be used as the basis for comparison for every data window. We refer to them as a *baseline group*.

The result of using a baseline group is that each epoch will essentially be represented by multiple signatures, one for each baseline group member. Another way to

Baseline	**Signature s_t**						**Signature $s_{t'}$**					**Cityblock Distance**
	m_1	m_2	m_3	m_4	m_5		m_1	m_2	m_3	m_4	m_5	
b_1	1	0	1	1	0		1	1	1	0	0	**2**
b_2	1	1	0	0	0		1	1	0	0	0	**0**
b_3	0	1	1	0	0		0	1	1	0	1	**1**
												Average Distance: 1.0

Table 7.1: **Examples of signatures for epochs t and t' using a baseline group consisting of 3 individual baselines $\{b_1, b_2, b_3\}$.** Each signature basically consists of sub-signatures, one for each baseline. Overall distance between signatures is computed by averaging the distance between sub-signatures.

think about this is that each epoch's signature will actually contain a vector of sub-signature vectors. These sub-signature vectors are what we previously represented as a signature. Distance between epochs is simply the average of distances between sub-signature vectors. See Table 7.1 for an example.

A *baseline group* could be any set of good behaviors. Each individual baseline could be any group of generally contiguous SLO compliance epochs. For example, the requirement for a baseline of length 100 might be that at least 95 (this is configurable) of these epochs must be compliance. Any violating epochs in between is simply ignored and the length of the baseline refers to the timespan between the first compliance epoch and the last compliance epoch used. The generally continuous requirement is due to the fact that changes in behavior become more likely over longer periods. There are two main parameters that govern the composition of a baseline group: the size of the group (i.e. number of good behaviors in the group), and the length of each baseline within the group (we require this to be uniform within a group). For example, a baseline group may consist of 5 baselines of 300 epochs each or perhaps 10 baselines of 100 epochs each.

The hope is that by using a baseline group, which represents multiple models of normality, we can at least achieve a consistent view of the symptoms of problems, regardless of the exact models of normality used. This is analogous to taking a patient's vital stats at many different times and using all of these measurements as the basis for symptom definition. While some measurements might be taken right after a long run or a large meal, considered together, they may represent a good, stable model of normality.

7.4 Evaluation

In this section, we aim to show that by utilizing baseline groups, we can better guarantee signature accuracy. We characterize the retrieval performance of signatures generated using randomly formed baseline groups on the labeled `IDC` problem from `FT-TRACE`. Analysis using the `TESTBED` data was excluded because all of the good behaviors in that data set were designed to be very similar. Thus it is not suitable for demonstrating the performance of our method.

We compare how the range of retrieval accuracies differ depending on the parameters for a baseline group, with using a group of size 1 meaning that we rely on a single baseline. Basically, histograms like those presented in Figure 7.1 have been summarized to only a mean and a standard deviation and are shown in Figure 7.2. Each pair of $\langle$mean, standard deviation$\rangle$ values represents 25 trials of forming a baseline group by randomly selecting good behaviors according to the given parameters.

The general pattern is clear. Using larger baseline groups (i.e. more baselines in a group), and to a lesser degree using longer baselines within a group, substantially increases the mean accuracy and decreases the variance. For example, using a randomly selected single baseline of length 300 yields an average accuracy of 0.77 and a standard deviation of 0.13. This implies that we can only guarantee an accuracy of 0.51 with about 88% certainty[1]. When using a baseline group of any 5 randomly selected baselines of length 300 yields, the mean accuracy increases to 0.90 while the standard deviation decreases to only 0.04. This guarantees an accuracy of 0.82 with the same certainty as before, a drastic improvement. We note that frequently, a group of baselines used together will result in more accurate signatures than any of the baselines used individually. This suggests that even if labels were available for evaluating individual baseline accuracies, using baseline groups would still offer an

[1]We calculate this using Chebyshev's inequality, which states that no more than $1/k^2$ of the values of a distribution are more than k standard deviations from the mean. This theorem valid for all statistical distributions while the common 95% within 2 standard deviation rule is only for normal distributions.

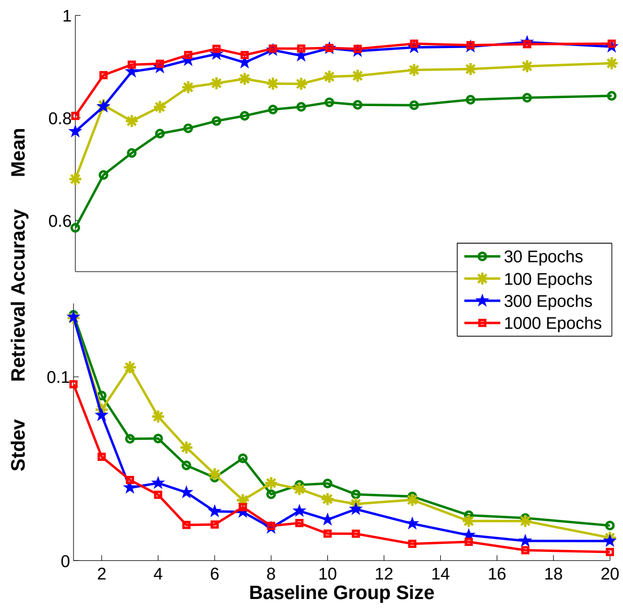


Figure 7.2: **Mean and standard deviation of retrieval accuracy against baseline group parameters.** Baseline groups were repeatedly generated using various combinations of the baseline group size (X-axis) and baseline length parameters. Signatures were created based on those baseline groups and evaluated against the IDC problem of FT-TRACE. Using baseline groups ensures better accuracy compared to using a single baseline, with larger groups offering the best improvement.

advantage. Therefore, not only do baseline groups result in more *consistent* signature retrieval results, but retrieval accuracy is also drastically improved.

However, there is a price to be paid for using larger baseline groups and longer baselines. Computation time for signature generation and retrieval (and clustering) is linear in the size of a baseline group. For longer baselines, there is extra computational cost for the signature generation phase only and it is again linear with respect to the number of epochs in each baseline. In addition, there are fewer possible long baselines since long periods of compliance without notable problems are infrequent in many systems.

Note how the marginal improvement in signature accuracy (both mean and standard deviation) diminishes for larger baseline groups and longer baselines. For this evaluation data, using a baseline group consisting of 5 baselines of 300 epochs each represents a reasonable balance of increased computational costs versus improved signature accuracy. However, this particular setting of the baseline group parameter may only be a good balance for this data set and possibly only the `IDC` particular problem. To continue with our analogy, while we may only require observation during 3 or 4 different periods of normality to achieve consistent symptom definitions for certain illnesses, more sets of observations may be needed for very complicated conditions. The next section presents a method of evaluating the consistency of signatures in the absence of labeled data.

7.4.1 Inconsistency

Recall that a main reason baseline groups are used is because evaluating individual baselines requires reliably labeled data. To solve the problem of optimizing the baseline group parameter setting in the absence of labeled data, we introduce a new measure, signature *inconsistency*. Inconsistency attempts to capture the variance of signature-based problem similarity inferences (for clustering or retrieval) constructed using different baseline groups (or different individual baselines), without knowing the accuracy of those inferences. This is somewhat analogous to making sure that a pair of patients is always deemed to have either the same set of symptoms or a different set of symptoms, regardless of whether or not they actually have the same illness (which is unknown).

We aim to utilize this inconsistency measure to determine how many different baselines must be incorporated in a baseline group to guarantee some level of consistency in signature-based inferences. We wish to show that inconsistency can serve as a *proxy* for the accuracy of signature retrieval. Recall that signature-based problem similarity inferences are determined by distances between signatures. Let S^A denote the set of signatures extracted from some dataset using the baseline group A. In particular, $S^A = \{s_1^A, ..., s_n^A\}$ where s_t^A is the signature of epoch t with respect to baseline group A. Similarly, we define S^B as the set of signatures extract from that dataset using baseline group B. We define the inconsistency of baseline groups A and

B on this dataset as

$$I(A, B) = \frac{1}{n^2} \sum_{t=1, t'=1}^{n,n} |D(s_t^A, s_{t'}^A) - D(s_t^B, s_{t'}^B)| \tag{7.1}$$

where $D(s, s')$ represents the distance between two signatures. Intuitively, this is the average difference in signature distance, over all pairs of epochs, when using one baseline group compared to a different group. To reduce computation time, we may sample pairs of epochs and average those results instead of using all possible pairs.

Figure 7.3 shows the inconsistency of using different baseline group parameters on the `FT-TRACE` dataset. Each point in the graph represents the average pairwise inconsistency of selecting two randomly formed baseline groups with those parameters. For example, the inconsistency value for baseline groups of size 5 and length 300 is determined by randomly forming two baseline groups, each containing 5 individual baselines of 300 epochs each, measuring the inconsistency of the signatures generated by using these two different baseline groups, and repeating this procedure at least 25 times and averaging the results. Note the similarity between this graph and the graph of the standard deviation of retrieval accuracy show in Figure 7.2. Indeed, Figure 7.4 shows the very strong correlation between these two measurements. Furthermore, since no labeled information is required for calculating inconsistency, we are not restricted to evaluation based only on the `IDC` signatures.

We therefore conclude that inconsistency can be used as a proxy of retrieval accuracy when annotations are unreliable or unavailable. However, there are important distinctions between the two measures. First, the numerical value of inconsistency is meaningless since it is not calibrated and depends on the exact distance metric employed. More importantly, low inconsistency implies low variance in retrieval accuracy, not good accuracy itself. However, low inconsistency is a necessary condition to being able to guarantee quality signatures when direct evaluation with labeled data is not possible. Intuitively, low inconsistency means that signature accuracy does not depend on the specific baselines used in the baseline group.

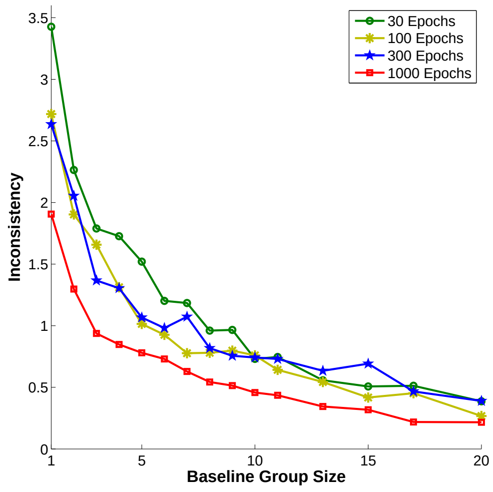


Figure 7.3: **Inconsistency against baseline group parameters.** Inconsistency (Y-axis) is calculated based on the differences in signature when based on a particular baseline group versus another baseline group with the same parameters. Labels are not used for this calculations so all signatures of FT-TRACE were considered instead of focusing on the IDC problem signatures only. Note that using baseline groups drastically decreases inconsistency, with the larger groups experiencing the lowest inconsistency.

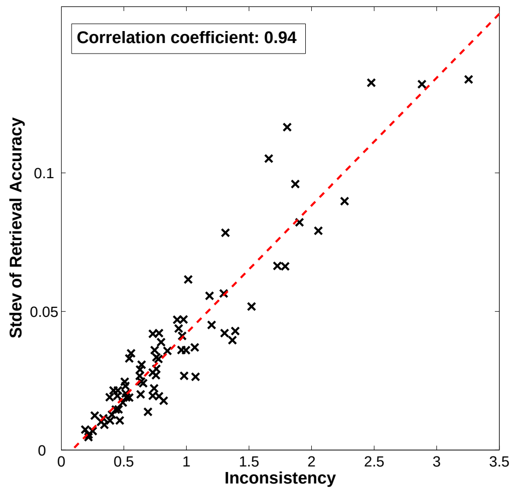


Figure 7.4: **Inconsistency versus standard deviation of retrieval accuracy.** When accurate labels for evaluation are unavailable, the inconsistency measure (Y-axis) can be used to directly assess the extent to which the accuracy of signature based inference various depending on the exact baselines chosen.

7.4.2 Practical implications

Operators can leverage this approach in a production environment as follows. Using existing system data, compute the inconsistency of using different baseline group parameters to generate a graph similar to Figure 7.3. Operators may then use this information to balance the increased computational costs of larger baseline groups versus improved consistency. Finally, once the baseline group parameters have been decided, choose a *single* baseline group using those parameters as the basis of all future signature construction. Due to the computationally intensive process of exploring consistency over different baseline group parameters, this procedure is designed to be used only during the initial deployment of our signature based techniques to a system. Once labeled problem instances are known, they can be used to directly assess the accuracy of generated signatures.

7.5 Summary

This chapter explored the implications of the observation that desirable system behavior (SLO compliance) does not usually exhibit a single monolithic pattern.

- We showed that using a *baseline group*, multiple baselines used in conjunction, results in good retrieval accuracy on our dataset no matter which individual baselines are used. We evaluated this claim using the `IDC` problem instances in `FT-TRACE`. We discovered that using a baseline group of size 5 could improve the lower bound of an 88% confidence interval of signature accuracy from 0.51 to 0.82.

- The composition of a baseline group is controlled by two main parameters: the size of the baseline group (how many individual baselines are used) and the length (number of epochs) of each baseline. Because larger groups and longer baselines result in significant additional computational costs and the marginal benefits of increasing group size decreases for larger groups, we must balance these tradeoffs when choosing a baseline group.

- In order to set the baseline group parameters wisely in the absence of reliable labels, we introduced the concept of inconsistency, which measures how much inferences based on signature distances can change due to the particular baseline group used. This measure can be used by operators to select baseline group parameters that balances computational costs against improved signature consistency. Although consistency is not a guarantee of good accuracy, it is necessary for instilling confidence in signature quality.

- Operators can periodically evaluate the accuracy of signatures with labeled problem instances. In addition, inconsistency can also be computed from time to time to gauge baseline group effectiveness in the absence of labels.

Chapter 8

Practical considerations and limitations

Our signatures based approach is well suited towards diagnosing problems in complex networked systems today. Often, although enormous amounts of data are collected about the behavior of a system, important information is usually difficult to uncover. Our statistical techniques are fast enough to extract useful patterns from typically overwhelming amounts of data and yet do not rely on *a priori* knowledge about system structure, which is difficult for operators to provide. However, our approach is only one key piece in the larger puzzle of the management of complex systems. In this chapter, we describe how operators can best leverage our diagnosis tool and the limitations of the approach.

8.1 Interaction with operators

There has been much recent progress towards automated system problem detection (such as Pinpoint [11]) and resolution (such as the microreboot approach described in [9]). Although some issues could be detected and resolved completely without operator support, many failures still need to be handled by human administrators. Therefore, we consider how operators should best leverage our tools and what kinds of human input can be fed back into these tools to maximize their usefulness.

8.1.1 Usage guide

An usage guide to our technique designed for system operators would include the following key points.

- **Data collection:** In general, the more data (both in terms of the number of metrics and in terms of more frequent reporting) collected about the system the better. As much should be captured about the system as possible without unacceptably degrading system performance. Monitoring processes should report information at least every few minutes. The faster that a system behavior may change, the shorter reporting epochs should be.

- **Signature clustering:** The clustering analysis should generally be used when a large set of signatures are unlabeled. This will provide recurrence and intensity information helpful towards prioritization of operator efforts. When utilizing k-means clustering, it is important to cluster using several different settings for k to ensure the stability of results.

- **Signature retrieval:** When problems occur as a system runs, the signatures for those problems can be used to retrieve similar problems in the past. This is most useful when many past problems have already been annotated with diagnostic information.

- **Using metric attribution:** Our suggested signature composition, based on metric attribution information, directly contains possible hints of the causes and effects of a system problem. The set of metrics deemed attributed represent a good starting point for diagnostic efforts.

- **Annotating signatures:** Any information helpful towards diagnosing or resolving a problem should be added to the appropriate signatures' annotations. Annotations can be altered at any time with no impact on signatures. Neither the signature clustering or retrieval process depends on the existence or accuracy of labels. However, the more accurate and relevant the information included with annotations, the more useful these tools will be.

- **Incorporating operator knowledge in the modeling process:** One of the reasons we advocate using Bayesian network classifiers is for the ability to inject expert knowledge into these models seamlessly. In fact, in the simplest usage cases, Bayesian networks can be completely specified by a human and then directly used for inference. A network consists of a set of dependencies between features (if any) and the conditional probability of each feature given its parent feature(s). We do not delve into the details of the process, but it is covered by most books on statistical and machine learning, such as [18].
- **Verification of signatures:** It is possible that signatures may provide inaccurate inferences in some systems or for some types of failures. Therefore, we suggest that operators verify the accuracy of signatures using already diagnosed failures in their system in a similar manner to how we evaluated our techniques in Chapter 5. If signature quality is found to be poor, there are many parameters that could be tuned in an effort to improve accuracy, as we discuss in section 8.1.3.

8.1.2 Verification

We demonstrated the excellent retrieval and clustering properties of metric attribution based signatures on two systems. However, this same approach might not work well on some given system for a wide variety of reasons. Perhaps some of the assumptions made in Section 2.2 do not hold; for example, important information relevant to overall system state may not have been monitored. Or perhaps our technique is simply not adapting fast enough to ephemeral behavior. Some of these issues could be resolved or mitigated by simply changing configuration parameters such as the training window size for the ensemble method (see Chapter 6). Therefore, the ability of an operator to accurately evaluate the quality of generated signatures is quite valuable not only to increase operator trust in the inferences made, but also because it is sometimes possible to remedy poor signature quality in a quick and effective manner.

The signature retrieval and clustering evaluations we described in Chapter 5 could also be applied by operators for their own systems. When reliable labels are available, we advocate focusing on validating good retrieval properties using precision-recall graphs. This is preferred over clustering for two reasons. First, retrieval measures signature quality more directly because clustering relies on the choice of a clustering algorithm as well as a method of optimizing the number of clusters (for k-means). In addition, clustering evaluation is most valuable when almost all violation signatures have already been labeled or annotated. Generating a precision-recall graph for a set of signatures is possible as long as most instances of a particular problem is labeled (as was the case for `FT-TRACE` and the `IDC` issue). Root cause diagnosis or resolution of the problem is not required. Knowledge that a particular subset of the signatures reflects a common problem would be sufficient.

When labels are unavailable, we can evaluate signatures by measuring clustering purity using the ratio of compliance versus violation signatures in each cluster. An average weighted entropy near zero would still firmly suggest that meaningful differences between desirable and undesirable system behavior are being accurately captured by the signatures. In addition, recall from Section 7.4.1 that the inconsistency measurement can be used to assess the stability of retrieval results. Although good (i.e. low) inconsistency does not necessarily imply accurate signatures, it is a necessary condition for robust signatures.

8.1.3 Troubleshooting signatures

We now describe a few cases where signatures may exhibit poor retrieval or clustering behavior and what remedies an operator might apply in response to those symptoms.

- **Very few or no metrics are deemed attributable for most SLO violation epochs.** There are a few possible explanations for this behavior. First, it is possible that our models are rarely confident that a metric value strongly signals a state of SLO violation. Recall that we use a threshold on the log-likelihood ratios to determine attribution. Theoretically, this threshold can be set as low as zero, meaning that as long as a metric's value indicates violation over compliance by any amount, it would be deemed attributable. Lowering this threshold from our suggested default of around 5 should result in more metrics being attributed.

If even lowering the log-likelihood ratio to zero still results in few attributions, then it means that very few accurate models are being induced. This may be due to a lack of correlations or patterns in the data itself (i.e. relevant metrics were not captured or perhaps not frequently enough), which may be solved by enabling additional logging or monitoring processes. However, it may also be because of an inability of our models to capture the types of patterns present in the data. More powerful classifiers and induction methods may be needed. Another factor may be the way that the SLO is defined, which will be discussed in Section 8.2.1.

- **Poor clustering and retrieval performance despite many attributions for SLO violations.** There are many different reasons this could be occurring. One possibility is that our models may be overfitting the data, extracting correlations that do not generalize well and do not reflect true system behavioral patterns. This could be addressed by using simpler models or induction algorithms.

  Another possibility is that the labels used for evaluation could be inaccurate. Operators are rarely very confident about being able to identify all of the instances of any particular problem. If the metrics deemed relevant (information contained in each signature) make sense intuitively from a systems point of view, then inaccurate ground truth is a possible culprit for poor evaluation results.

- **Retrieval accuracy is poor, but clustering does a good job of separating violation and compliance.** This suggests that although metric attribution is capturing relevant metrics that correlate well with overall system behavior, information that would separate different types of violation behaviors is not being captured or represented. The feature selection process could be partially responsible for this problem and may be tuned to filter out fewer metrics (at the risk of causing overfitting).

Note that these are not the only problems that our approach might suffer nor are these the only methods that may mitigate the problem. However, we believe that these are the most common issues that a user of our approach might encounter.

8.2 Limitations

As systems become ever more complex and monitoring increases, the need for automated diagnostic techniques will only grow. However, signatures will not be able to address every problem. This section will explore some limitations of not only our approach but also some that are fundamental to any automated method.

8.2.1 SLO Definitions

Although we advocate for the general case of using signatures to identify similarity between problems, it is clear from this thesis that the metric attribution technique for generating signature offers much promise. However, this approach is also subject to the pitfalls of relying on SLO definitions, which may be set based on business needs rather than normal system behavior under typical workloads. The lack of constraints on how a service level objective can be defined is also a great advantage of this technique and will be discussed further in Section 10.3.

SLO definitions can be based on any number of metrics that summarize overall system performance. These include but are not limited to performance based measures such as latency and throughput, reliability based metrics such as the mean time to failure, and other measures such as click through rates or even survey based user feedback. Although these high level measures are often easy to monitor and report, not much effort may be put into finding "good" thresholds (that truly differentiates desirable versus undesirable behaviors). Contractual agreements between executives often drive these decisions, rather than systems reasoning. For example, most systems have a limit on the amount of input they can handle before they either slows to a crawl, crash, or rate-limiting pre-processing has to be applied. Even though latency may naturally increase as workload ramps up, there usually comes to a point where this latency increase is no longer only due to processing delays. Components may stop responding, causing numerous transactions to time out. Ideally, an SLO threshold based on response times should factor in this aspect of system behavior.

To maximize the potential benefits of our approach, we advocate that if possible, SLO threshold be determined through technical evaluation of system behavior. For example, if average transactional response time is used (as it most often is), the actual threshold should be set only after analyzing the typical response times of the system under normal workload conditions.

One key limitation of our approach is that very infrequent SLO violations will be particularly difficult to model. As shown in Section 6.3.2, at least 20 or 30 violation epochs are needed to robustly characterize even simple problem behaviors. When this occurs, an operator may choose to use a more stringent "virtual" SLO for signature generation purposes only. In this manner, signature will characterize not only true SLO violating behavior, but also behavior close to the SLO threshold. Such information may prove to be valuable for preventing true SLO violations.

In general, a major issue with SLOs is that there is no guarantee that there is actually meaningful differences between SLO states. Suppose a system usually exhibits average response time around two seconds under normal workloads and the SLO threshold were also set to around 2 seconds, violations will be frequent and usually be just above the threshold, while compliance epochs tend to fall just barely below the threshold. There would be no meaningful correlations detected in such a case.

A related issue was seen in our `FT-TRACE` data set (Figure 8.1). While the threshold itself of 4 seconds is reasonable, the reported average response time occasionally oscillated wildly, from less than 2 seconds to more than 8 seconds every few epochs for several hours. Although it is possible that system behavior is truly changing that rapidly, most operators would simply consider the behavior of this entire period to be undesirable[1]. However, because the SLO is based on individual 5 minute epochs, half of these epochs would be considered compliant behavior. Smoothing the SLO

[1]In fact, we believe that the underlying problem causing these violations persisted throughout that time range and the oscillations in response time were due to an artifact of how response times of failed transaction are considered.

so that compliance or violation is determined by more than just the current epoch can help mitigate this problem. We applied rudimentary smoothing to the `FT-TRACE` data and discovered that doing so led to more accurate modeling of system behavior, especially during these oscillatory periods.

8.2.2 Root cause diagnosis

The statistical and pattern recognition techniques underlying the automated extraction of signatures capture correlation, not necessarily causation. Indeed, as is well known in statistics and in other communities, the ability to infer causation from pure observation is limited and in most cases impossible [42]. By pure observation we mean lack of direct intervention into the system or additional information from human experts regarding the causal relations and paths in the system. In some instances, time information and information about the sequence of events can be used as heuristics to find causal connections. This has been attempted in many domains including this one, most notably in [1]. We leave as future work the inclusion of this kind of information into our approach and the exploration of its utility, although we remark that there is nothing in principle that prevents us from considering sequences of signatures or adding time information (including precedence information) into the creation of the signatures and the subsequent analysis. See Section 10.1 for further discussion on this topic.

It follows from this limitation that we cannot and have not claimed that the approach advocated in this dissertation yields the root causes of problems. Even with human expert knowledge, root cause analysis is far from trivial. Nevertheless, we believe that the capability of systematic similarity search and clustering of correlated metrics offered by signatures helps in narrowing down possible causes and is therefore useful as a diagnosis tool. Furthermore, it may not be necessary to determine exact root causes to resolve problems. As difficult as root causes analysis has proven to be over the years, a more pragmatic approach could be to automatically map the evidence for faults and metric state to a finite set of possible repair actions.

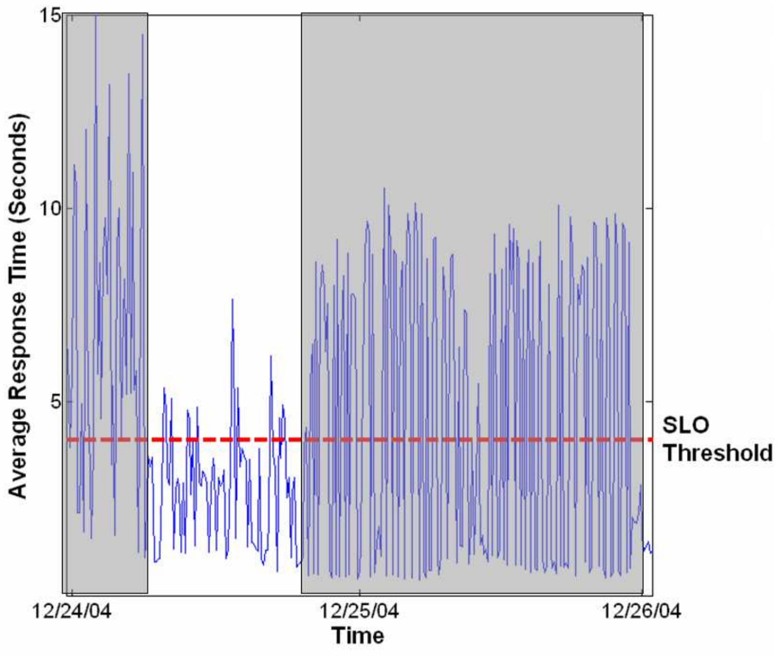


Figure 8.1: **Example of SLO state not representative of system behavior.** Although only about half of the epochs during this time period violate the SLO threshold of 4 seconds, it is most likely that almost this entire period's behavior is undesirable. Simple smoothing techniques can be applied to mitigate this problem. After smoothing this data, the shaded regions were all considered violations and with all compliance in between. This resulted in modeling accuracy of this period increasing from the low 61% range to 92%.

Chapter 9

Related work

Systems today are becoming increasingly complex and difficult to manage. Current tools that assist operators for detecting or diagnosis problems are quite primitive. Therefore, it is not surprising that there have been a significant amount of work done in just the last few years in this general domain. This chapter will first talk about statistical and non-statistical techniques for diagnosing system failures. It then explores statistical techniques for the detection of problems as well as signature based approaches outside of system problem diagnosis.

9.1 Techniques for system problem diagnosis

There have been several projects in just the last few years focusing on diagnosing systems issues using statistical approach. First, at HP Labs, Aguilera *et al.* describe two algorithms for isolating performance bottlenecks in distributed systems of opaque software components [1]. Since many systems today use off the shelf components from third party vendors, operators of these system have no way of instrumenting or examining the source code of these components. Thus, each entity must be treated as a *black box*. Often, the only information that is readily available is the timing information about communications between the black box components. The two algorithms described in this work were a convolution algorithm and a nesting algorithm. Both algorithms seek to infer causal connections between input and output communications of a component, which leads to average latency information. The convolution algorithm makes no additional assumptions and can only be used to compute performance information for the majority behavior of the system. The nesting algorithm assumes RPC style messaging semantics and is able to isolate the behavior of specific requests or transactions.

At the opposite extreme of that knowledge-lean approach, Magpie, developed at Microsoft Research Cambridge, characterizes transaction resource footprints in fine detail but requires that application logic be meticulously encoded in “event schema” [2]. The goal is to extract representative models of resource usage and behavior for system performance modeling and debugging. Detailed instrumentation at the operating system level tracks specific resource usage characteristics for transactions, which are translated by the event schema to form a very fine grained runtime path model. Magpie then uses data clustering techniques to determine runtime paths that succinctly model a system’s overall behavior.

Both works focus on performance debugging and identifying bottlenecks in systems. Neither advocates a signatured based approach for identifying similarity between problems or is suitable for problems not tied to performance as measured by latencies. Chen, Zheng *et al.* at U.C. Berkeley and Ebay.com, utilize decision tree learning algorithms for fault localization [12]. Although similar to our approach in that they do not focus on fault detection but rather on diagnosing and localizing of the problem, their approach also differs in the lack of a signature based method.

Bodík, *et al.* examined how troubleshooting and monitoring was conducted at Amazon.com [3]. They found three main challenges.

- Complex dependencies in the system cause failures to propagate quickly, making root cause diagnosis difficult.
- Lack of global dependency knowledge. Each operator knows only a small part of the system well. No individual understands all of the behavioral dependencies between different parts of the system.
- Although the whole system is heavily instrumented at a detailed level, most

> problems can be characterized by the behaviors of tens of metrics. The total amount of information collected is overwhelming and makes analyzing problems extremely challenging for operators.

These observations strongly reinforce the motivations for our approach.

They also noted that system operators and troubleshooters generally use hardware, operating system, network, and application metrics that are collected from every machine at fine grained intervals. All of this data is stored in a central database accessed via web-based tools designed in-house. These tools are generally used to graph the metrics but are also capable of some simple analysis, such as predicting metric values based on historical values. Setting threshold based alarms for any metric is also possible. Different troubleshooting teams can design their own tools that operate on top of this environment. Sometimes, operators actually log into individual nodes to look at event and error logs or restart processes, although that usage is declining as the web-based tools are further developed.

As of November 2005, a new visualization tool called Maya was being tested at Amazon.com. Maya shows the dependencies between different system components as a directed graph. The health of each component is also displayed. Users may zoom into any node to examine the metrics of each components. Although Maya is certainly an improvement over previous methods, operators still felt overwhelmed about the amount of information available. A single component may have hundreds or thousands of metrics, and Maya offers no systematic way of identify which metrics are the most relevant. It is clear that more advanced tools are still needed.

Jain describes a traditional performance debugging technique that generates *visual* signatures of performance problems [24]. Popular in the 1970s, "Kiviat graphs" display a handful of utilization metrics in such a way that resource bottlenecks and imbalances assume a distinctive appearance. Like our approach of using signatures, Kiviat graphs allow comparison and facilitate similarity matching between problems in same or different systems. However, rather than relying on visual inspection, our approach allows for *automated* indexing, retrieval, and similarity measurement, which scales better for thousands or millions of metrics in the complex systems of today.

There have been a significant amount of literature on *faults*, rather than performance problems, in distributed systems. For communication networks, whose components have highly constrained and well defined specific behaviors, a wide of *fault localization* techniques have been explored [48]. Yemini *et al.* describe an *event correlation* (root cause determination) procedure that relies on an extensive library describing each system components' possible faults and the consequences of each fault [52]. These detailed component descriptions are compiled into a *codebook* that reduces root cause analysis to a simple and efficient task of decoding observed symptoms into the faults that caused them. This approach has been commercialized for communication systems [23] but is inappropriate for arbitrary distributed software systems because it is infeasible to enumerate in advance the faults and symptoms of arbitrary computer programs. In addition, this technique has no learning or adaptive aspects, as our approach does.

9.2 Statistical techniques for system problem detection

The first step necessary for resolving a failure is to detect it. While problems associated with violating SLOs are usually trivial to detect, many types of failures are subtle and difficult for operators to notice immediately. One example is failures in which a part of a service is providing incorrect output. Several recent projects utilize statistical anomaly detection to approach this problem.

The Pinpoint system of Chen *et al.* analyzes run-time execution paths of complex distributed applications. It automatically detects potential failures by identifying statistically abnormal paths [11]. These paths can then aid system administrators in diagnosing the underlying cause. Kıcıman & Fox further explore usage of Probabilistic Context-Free Grammars (PCFG), which are frequently used in natural language processing, in Pinpoint to model the typical behavior of a system's transactions [27]. The statistical likelihood of the path taken by any transaction can then be calculated with sound probabilistic reasoning. Because Pinpoint requires instrumenting components or middleware to obtain the transactional path information, it is ideally suited for Internet services that rely on an application server environment such as J2EE or Microsoft's .NET. Pinpoint also performs some fault localization using decision trees. At NEC Labs, Jiang *et al.* extended the approach taken by Pinpoint by using multi-resolution learned automata instead of PCFG path-shape analysis [25].

At U.C. Berkeley, Bodík *et al.* detected failures in Internet services by examining end user behavior patterns [4]. They use statistical analysis to find possible anomalies and visualizations to allow operators to easily identify the existence of a potential problem. In their analysis using HTTP logs from a real Internet service, they were able to automatically recognize several failures, often significantly faster than the in-place detection processes. Since user behavior is used and readily available as web server logs, no additional system instrumentation is usually required.

9.3 Statistical and signature based approaches in other domains

Signatures have been used extensively in virus scanning and intrusion detection [39]. Statistical techniques are often employed to flag anomalous activity automatically, but signatures of malicious behavior are almost always defined manually. Kephart *et al.* describe a statistical method for automatically extracting virus signatures for a commercial detection product [26].

Engler *et al.* use pattern matching techniques for bug discovery [19]. Based on the source code of a large system, their approach infers correctness rules such as function-call ordering constraints and locking requirements for critical sections. Once the rules are discovered, they can utilize them to find instances in the source code where those rules are not followed. Kremenek *et al.* continued this work by incorporating statistical methods to reduce false positives when locating bugs in system source code [31, 30]. Liblit *et al.* advocate *statistical debugging* [33]. With this technique, they sample code-level information, such as function return values, of software during normal execution. They use statistical analysis to find what information is most correlated with the symptoms of Heisenbugs, helping software developers locate bugs' root causes.

There have also been a large body of work incorporating statistical learning for optimization of the configurations of complex systems or parts of systems. David Sullivan applied Bayesian networks for software configuration and performance tuning in database systems [49]. He designed a controller that could automatically adapt a database system's configuration to changing workloads. Mesnier *et al.* explored the usage of decision trees to classify files and select storage policies to maximize the performance of file systems [36]. Finally, IBM has an initiative called autonomic computing, which seeks to design self-managing and self-configuring systems using control theory and some statistical techniques[40].

Chapter 10

Future work

A plethora of avenues remains for future research based on statistical techniques for detecting, diagnosing, and resolving system problems. More specific to this thesis, there are many opportunities for extending our approach of extracting signatures of system state to aid diagnosis efforts.

We explore three possible extensions. First, many problems manifest not as a single major shift in system behavior but rather as a series of subtle changes. These types of issues could be better captured in signatures if we were able to model temporal patterns in the monitored data. Also, due to recent work towards automated detection and repair methods, it is also not inconceivable that an automated diagnostic system could be combined with other automated tools to drastically reduce the troubleshooting role of a system administrator. Finally, although we validated our techniques on three-tiered Internet services, the generality of our approach leads us to believe that it can be effectively applied to a wide range of computer systems and possibly even outside this domain. We now discuss each of these future research directions.

10.1 Detecting temporal patterns

A memory leak usually results in a sometimes slow but steady increase in memory utilization, rather than a sudden jump when the problem is first triggered. Misbehavior in a single component often causes other components to fail over time, usually due to complex dependency chains. Network activity tends to ebb and flow frequently and suspicious behavior is often reflected by a change in this frequency. What these types of problems all have in common is that they are best described by a *sequence* of changes to system behavior.

While our current methods may find some of these correlations without directly considering time, many important patterns will be missed. There are two general ways of extending our approach to accommodate temporal patterns. The first approach is simpler but less powerful, while the second would require significant changes to the general signature framework but would be able to better capture many behaviors.

First, metrics may be manipulated such that while sequences of metric vectors is not directly used, some simple time based patterns can still be extracted. Instead of a vector of system data including only information from one epoch, it could also include metric values from past epochs, as well as the rate of change of those values. For example, if m_t is the value of some metric m at time t, then the data analyzed by our technique for each epoch may also consider m_{t-1}, m_{t-2}, $(m_t - m_{t-1})$, and $(m_t - m_{t-2})$. This approach requires only a simple alteration of the data preprocessing step, rather than fundamental changes to our techniques. However, many temporal correlations cannot be represented in this manner. In addition, adding many additional metrics for each existing metric can worsen the dimensionality problem and increase computation costs.

The second general method of capturing temporal patterns would involve explicitly searching for patterns in sequences of metrics' values. The simple Bayesian network models we currently employ would not be adequate for this task. Inducing robust models that capture temporal based patterns is quite difficult. However, there has been much research into this area. Hidden Markov models (HMMs) have been used successfully to model sequence based information [13, 34]. Many have also studied pattern extraction from financial time series events (e.g. stock prices) [43]. The key for adapting those approaches in the domain of system problem diagnosis is to use interpretable models so that the captured patterns can be presented to operators and summarized in signatures.

10.2 Automated diagnosis and repair

Although fully automated system management may not be realistic in the near term, many simpler types of failures could be resolved without operator intervention. One caveat, however, is the pitfall of automation irony [44], which states that automation increases system complexity and hinders operator understanding of a system to the point that operators become ill-equipped to resolve problems with the automated process itself. Thus it is important for automated tools to be transparent to users and their actions deterministic and easy to understand. Simple, low cost[1] resolution methods such as microrebooting components [8] fit such a profile.

Our signatures based approach may be used to aid automated repair in several ways. First, attributed metrics may be examined to extract a list of components implicated in a failure. That list of components could then be microrebooted (or some other resolution method applied). Another approach could involve simply mapping signatures to repair actions. Since signatures are effective at identifying similarity between problems, the proper repair actions could be attached to signatures such that whenever a new problem is determined to be similar to a prior one, the previous repair action is automatically applied.

10.3 Application to different systems and domains

While we validated our approach using performance based problems in three-tiered Internet services, a signature based method involving correlations between copious low level data and some high level performance measure is not restricted to three tiered Internet services or even to large networked systems. Single machines may also

[1]Low cost means that the penalty for doing the resolution action unnecessarily is low.

exhibit complex behavior patterns that prove different for operators to characterize. An important aspect of future work in this area will be to explore the generality of our techniques. Our methods may need to be adapted to be effective for vastly different types of systems or different types of failures.

Our approach may even be applicable to outside the domain of computer systems. Many general problems fit the profile of correlating low level properties with some high level metric. For example, statistical and machine learning algorithms have been successfully applied to the credit risk assessment process that banks use to determine whether or not to grant a loan (based on credit history, income, education, and a wide array of other factors), although in that case, the classification prediction itself is what is most important to the users of the technique. However, our signature based approach allows representation of the robust patterns captured by good classifiers and how those patterns may change over time. We believe our techniques demonstrate a framework for extracting and leveraging such information, which may prove valuable in many situations.

Chapter 11

Conclusions

This dissertation attacks the problem of using automatically-computed statistical digest representations to diagnose problems in complex systems. We advocate a signatured based approach that captures essential system state in an indexable form.

We demonstrated the efficacy of a statistical machine learning technique as the basis for signature generation. A statistical approach is able not only to analyze enormous amounts of data about observed behavior, but also to quickly adapt to system changes, a major advantage over techniques that require *a priori* knowledge of system structure. We showed the ability of signatures to accurately identify similarity between performance based problem instances in two traces, one from a controlled experimental testbed system with three injected failures, and the other from a globally distributed production environment that experienced several problems over a month-long period. Signatures aid system operators by offering a systematic way of leveraging past diagnosis efforts for future problems. In addition, the clustering of signatures offers a summary of the different types of failures that have affected the system and how frequently each occurs, allowing administrators to prioritize their efforts. In effect, we are able to cast diagnosis as an information retrieval task, allowing well-known techniques from that domain to be incorporated.

We also addressed challenges that not only impacted our approach but also have bearing on other statistical approaches for system diagnosis. We introduced a method for adapting statistical models to rapidly evolving system behavior using an ensemble of models. Also, we characterized the dynamic nature of not only failures but also desirable system states and developed a technique for making signatures more robust to this variance.

We believe the success of our approach makes it a promising start for leveraging statistical and information retrieval techniques to address the challenges posed by the complexity of today's and tomorrow's systems.

Bibliography

[1] Marcos K. Aguilera, Jeffrey C. Mogul, Janet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proc. 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, 2003.

[2] Paul Barham, Rebecca Isaacs, Richard Mortier, and Dushyanth Narayanan. Magpie: real-time modelling and performance-aware systems. In *Proc. 9th Workshop on Hot Topics in Operating Systems*, Lihue, Hawaii, June 2003.

[3] Peter Bodik, Armando Fox, Michael I. Jordan, David Patterson, Ajit Banerjee, Ramesh Jagannathan, Tina Su, Shivaraj Tenginakai, Ben Turner, and Jon Ingalls. Advanced tools for operators at amazon.com. In *First Workshop on Hot Topics in Autonomic Computing (HotAC'06)*, Dublin, Ireland, June 2006.

[4] Peter Bodík, Greg Friedman, Lukas Biewald, Helen Levine, George Candea, Kayur Patel, Gilman Tolle, Jon Hui, Armando Fox, Michael I. Jordan, and David Patternson. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In *Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC '05)*, June 2005.

[5] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[6] Eric Brewer. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, July 2001.

[7] G.W. Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3, 1950.

[8] George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, and Armando Fox. A microrebootable system – design, implementation, and evaluation. In *Proc. 6th USENIX OSDI*, San Francisco, December 2004.

[9] George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, and Armando Fox. A microrebootable system - design, implementation, and evaluation. In *Proc. 6th USENIX OSDI*, December 2004.

[10] R. Caruana, Al. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *International conference on Machine learning ICML*, 2004.

[11] Mike Chen, Emre Kıcıman, Eugene Fratkin, Eric Brewer, and Armando Fox. Pinpoint: Problem determination in large, dynamic, internet services. In *Proc. International Conference of Dependable Systems and Networks*, pages 595–604, Washington, DC, June 2002.

[12] Mike Chen, Alice Zheng, Jim Lloyd, Michael Jordan, and Eric Brewer. A statistical learning approach to failure diagnosis. In *Proceedings of the First International Conference on Autonomic Computing*, May 2004.

[13] Darya Chudova and Padhraic Smyth. Sequential pattern discovery under a markov assumption. Technical report 02-08, Information and Computer Science Dept., University of California, Irvine, 2002.

[14] I. Cohen and M. Goldszmidt. Properties and benefits of calibrated classifiers. In *8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 125–136, September 2004.

[15] I. Cohen, M. Goldszmidt, T.P. Kelly, J. Symons, and J. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis

and control. In *6th Symposium on Operating Systems Design and Implementation (OSDI'04)*, December 2004.

[16] Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, indexing, clustering and retrieving system history. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP-16)*, October 2005.

[17] P. Domingos. Bayesian averaging of classifiers and the overfitting problem. In *International Conference on Machine Learning ICML*, pages 223–230, 2000.

[18] R Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1973.

[19] Dawson R. Engler, David Yu Chen, and Andy Chou. Bugs as inconsistent behavior: A general approach to inferring errors in system code. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 57–72, 2001.

[20] G. Forman and I. Cohen. Learning from little: Comparison of classifiers given little training. In *8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 161–172, September 2004.

[21] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[22] Hewlett-Packard. Management software: Hp openview, 2007. `http://openview.hp.com`.

[23] System Management Arts (SMARTS) Inc. Automating root causes analysis, 2001. `http://www.smarts.com`.

[24] Ravi Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley-Interscience, New York, NY, 1991.

[25] Guofei Jiang, Haifeng Chen, Christian Ungureanu, and Kenji Yoshihira. Multiresolution abnormal trace detection using varied-length n-grams and automata. In *Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC '05)*, June 2005.

[26] Jeffery O. Kephart and William C. Arnold. Signatures. In *Proc. 4th Virus Bulletin International Conference*, 1994. `http://www.research.ibm.com/antivirus/SciPapers/Kephart/VB94/vb94.html`.

[27] Emre Kıcıman and Armando Fox. Detecting application-level failures in component-based internet services. Submitted for publication, September 2004.

[28] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1137–1145, 1995.

[29] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[30] Ted Kremenek, Ken Ashcraft, Junfeng Yang, and Dawson Engler. Correlation exploitation in error ranking. In *SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, pages 83–93, New York, NY, 2004. ACM Press.

[31] Ted Kremenek and Dawson Engler. Z-ranking: Using statistical analysis to counter the impact of static analysis approximations. In *Proceedings of the 10th International Static Analysis Symposium*, June 2003.

[32] D. Richard Kuhn. Sources of failure in the public switched telephone network. *IEEE Computer*, 30(4), April 1997.

[33] Ben Liblit, Alex Aiken, Alice X. Zheng, and Michael I. Jordan. Bug isolation via remote program sampling. In *ACM SIGPLAN 2003 Conference on Programming Languages Design and Implementation*, June 2003.

[34] Dimitrios Makris and Time Ellis. Automatic learning of an activity-based semantic scene model. In *Proc. of IEEE Conference on Advanced Video and SIgnal Based Surveillance*, July 2003.

[35] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: Design, implementation and experience, 2003. `http://ganglia.sourceforge.net/`.

[36] Michael Mesnier, Eno Thereska, Gregory R. Ganger, Daniel Ellard, and Margo I. Seltzer. File classification in self-* storage systems. In *Proceedings of the 1st International Conference on Autonomic Computing*, pages 44–51, May 2004.

[37] Jeffrey C. Mogul. Emergent (mis)behavior vs. complex software systems. Technical report, HP Laboratories Palo Alto, 2005.

[38] David Mosberger and Tai Jin. httperf: A tool for measuring Web server performance. In *First Workshop on Internet Server Performance (WISP)*. HP Labs report HPL-98-61, June 1998.

[39] Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, May 1994.

[40] Sujay Parekh, Neha Gandhi, Joe Hellerstein, Dawn Tilbury, T. S. Jayram, and Joe Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Systems*, 23(1-2):127–141, July-September 2002.

[41] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[42] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.

[43] Richard J. Povinelli. Identifying temporal patterns for characterization and prediction of financial time series events. *Lecture Notes in Computer Science*, 2001.

[44] James Reason. *Managing the Risks of Organization Accidents.* Ashgate Publishing Limited, 1997.

[45] Reuters. Cbot suffers 3 electronic trading outages in 2 days. *Chicago Business*, January 2007. `http://chicagobusiness.com/cgi-bin/news.pl?id=23473`.

[46] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Mordern Approach.* Prentice Hall, 2nd edition, 2002.

[47] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[48] Malgorzata Steinder and Adarshpal S. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, (53):165–194, 2004.

[49] David Sullivan. *Using probabilistic reasoning to automate software tuning.* PhD thesis, Harvard University, Cambridge, MA, September 2003.

[50] The Open Group. Application Response Measurement (ARM) 2.0 Technical Standard, July 1998. `http://www.opengroup.org/onlinepubs/009619299/toc.pdf`.

[51] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Tecniques with Java Implementations.* Academic Press, 2000.

[52] Shaula A. Yemini, Shmuel Kliger, Eyal Mozes, Yechiam Yemini, and David Ohsie. High speed and robust event correlation. *IEEE Communications Magazine*, pages 82–90, May 1996.

[53] Steve Zhang, Ira Cohen, Moises Goldszmidt, Julie Symons, and Armando Fox. Ensembles of models for automated diagnosis of system performance problems. In *DSN*, 2005.

[54] Steve Zhang, Ira Cohen, Moises Goldszmidt, Julie Symons, and Armando Fox. Ensembles of models for automated diagnosis of system performance problems. In *The International Conference on Dependable Systems and Networks (PDS Track)*, Yokohama, Japan, July 2005.