Draco: Statistical Diagnosis of Chronic Problems in Large Distributed Systems

Soila P. Kavulya[†], Scott Daniels[§], Kaustubh Joshi[§], Matti Hiltunen[§], Rajeev Gandhi[†], Priya Narasimhan[†]
Carnegie Mellon University[†], AT&T Labs — Research[§]
{spertet,rgandhi,priyan}@ece.cmu.edu, {daniels,kaustubh,hiltunen}@research.att.com

***Abstract*—Chronics are recurrent problems that often fly under the radar of operations teams because they do not affect enough users or service invocations to set off alarm thresholds. In contrast with major outages that are rare, often have a single cause, and as a result are relatively easy to detect and diagnose quickly, chronic problems are elusive because they are often triggered by complex conditions, persist in a system for days or weeks, and coexist with other problems active at the same time. In this paper, we present Draco, a scalable engine to diagnose chronics that addresses these issues by using a "top-down" approach that starts by heuristically identifying user interactions that are likely to have failed, *e.g.*, dropped calls, and drills down to identify groups of properties that best explain the difference between failed and successful interactions by using a scalable Bayesian learner. We have deployed Draco in production for the VoIP operations of a major ISP. In addition to providing examples of chronics that Draco has helped identify, we show via a comprehensive evaluation on production data that Draco provided 97% coverage, had fewer than 4% false positives, and outperformed state-of-the-art diagnostic techniques by up to 56% for complex chronics.**

I. Introduction

The evolution of large distributed systems into entire *platforms* that provide dozens of distinct services to millions of users requires rethinking classic notions of availability as a binary property. Systems at such scales are rarely simply "up" or "down"; even when they are working for an overwhelming majority of users, there are almost always multiple ongoing problems of different types that affect small subsets of users. Often, the symptoms of each individual problem are not big enough to trigger alarm thresholds, and thus they fly under the radar of operations teams that are geared towards major outages. We call such problems *chronics*—small problems that persist in large distributed systems for days or even weeks before they are detected (often as a result of customer complaints). Chronics can occur repeatedly but unpredictably for short durations of time, or persist, affecting small subsets of users all the time. Nevertheless, they can cumulatively contribute significantly to the degradation of user experience. For example, data we obtained from the Voice over IP (VoIP) platform of a major ISP revealed that even in the worst month for major outages, the number of calls affected (dropped or blocked) due to major outages was only 30% higher than the number of calls impacted by chronics.

The discovery and diagnosis of never-before seen chronics in platforms comprising thousands of network, server, and user elements poses new challenges compared to the diagnosis of major system outages. Threshold-based techniques [4], [6], [15] do not work well because lowering thresholds to detect chronics often increases the number of false positives. Long-running persistent chronics can get absorbed into a system's definition of "normal", thus posing problems for methods based on historical models [9] or change-point detection [1]. Isolating individual problems is also more difficult—due to their persistent nature, lots of chronics are often present in a system at once, all starting and ending at different times, with larger problems hiding smaller ones. Furthermore, they occur even when the system works well for most users, and cannot be diagnosed by isolating the system's execution into periods of "good" and "bad" behavior [17], [20]. Finally, chronics often involve some unexpected combination of corner-cases that impact only small subsets of users, *e.g.*, a configuration error that impacts only those users with a particular version of a software stack, or a performance degradation that occurs only when the load on a particular server temporarily increases beyond a certain threshold.

To address these challenges, we present a new system called Draco[1] that can perform statistical diagnosis of chronics on large systems by combining data logs from different sources, and by diagnosing multiple ongoing problems—each identified by complex signatures across multiple dimensions. Draco can handle both discrete and real-valued data, and is threshold-free to allow detection of even small problems. To enable discovery of problems that have never been seen before or those that have persisted in the background for a long time, Draco does not rely on historical data. Draco minimizes false positives by using a "top-down" approach that relies on a scalable Bayesian distribution learner and an information-theoretic measure of distance (KL distances) [12] to identify sets of "problem signatures" that together explain the differences between the failed and successful user interactions.

Draco's core engine is scalable and domain-agnostic—requiring only changes to some well-isolated data parsers to be adapted to other applications. It is currently in production use by the operations team of a major US-based provider's VoIP platform that handles tens of millions calls per day. The problems we address, and our solutions, are not limited to VoIP. They are likely to be applicable to many other large platforms (*e.g.*, e-commerce, web-search, social networks) that serve users via independent interactions such as web requests.

[1]Draco is a genus of gliding lizards from Southeast Asia.

The contributions of this paper include the core algorithms for chronics diagnosis, Draco's scalable design, and a comprehensive evaluation of it's speed and accuracy. We evaluate Draco's quality of diagnosis in two ways: 1) through fault injection experiments that use real logs from our VoIP deployment, but inject a variety of precisely controlled synthetic failures so that ground truth is known, and 2) by cataloguing actual incidents on the VoIP network that Draco was able to identify, and which were subsequently confirmed by network operations personnel.

This paper extends our earlier work [10] by handling both discrete and real-valued data. In addition, this paper includes a comprehensive evaluation showing that Draco is able to quickly identify the attributes which are indicative of failure with a high level of up to 97% coverage, while maintaining a low level of less than 4% false positives. Draco also outperforms state-of-the-art diagnostic techniques that rely on decision trees [11], [20] by up to 56% when diagnosing complex problems involving multiple attributes while still providing near-interactive performance (< 1 second per problem) on large datasets.

The paper is organized as follows: Section II provides a brief background on VoIP networks and describes the VoIP dataset. Sections III and IV discuss the design and implementation of our diagnostic tool. Section V presents the results of our fault injection experiments, while Section VI presents case studies from production use where Draco helped in identifying the root-causes of chronic problems. Finally, Section VII compares Draco to related work.

II. Chronics in Telecom Systems

As of December 2010, 31 percent of the more than 87 million residential telephone subscriptions in the United States were provided by interconnected VoIP providers. In addition, approximately 31 percent of residential wireline 9-1-1 calls were made using VoIP services, making the availability of VoIP infrastructure critical [8].

We investigate chronics discovery for a part of the VoIP operations of a major US-based ISP. The portion of the ISP's VoIP network that we analyzed handles tens of millions of calls each day, contains several hundred network elements, and is layered on a large IP backbone. The network offers a portfolio of voice services including individual accounts, self-managed solutions where customers manage their own premise equipment (PBXs), and wholesale customers who buy network minutes in bulk and resell them. Calls traverse through combinations of network elements such as VoIP gateways (IPBEs), traditional phone gateways (GSXs), accounting servers, application servers, voicemail servers, and policy servers (PSX). Many of these are built by different vendors and have different log file formats.

To satisfy the high availability requirements of the system, there are real-time operations teams that monitor both low-level alarms derived from the equipment (server and network errors, CPU/memory/network utilization counters, *etc.*), as well as end-to-end indicators such as customer complaints

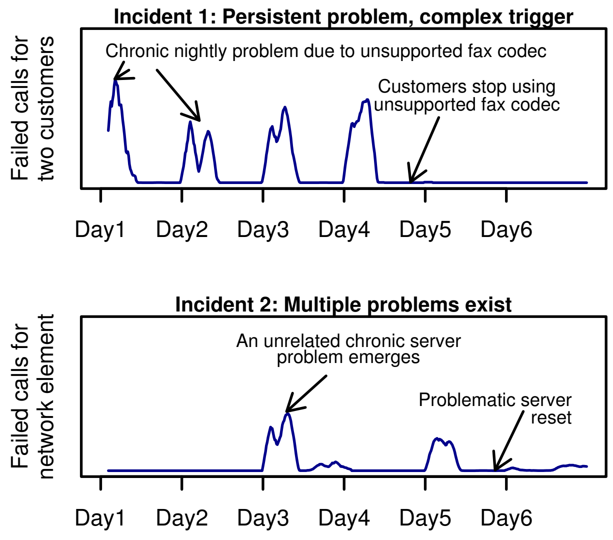


Fig. 1. Draco identified multiple ongoing problems that affected calls passing through the same network element at a production VoIP system.

and output from automated test call systems. Codebook-based systems [22] that are driven by signatures of known problems are used for identifying related alarms and for diagnosis. Major outages often result in immediate impact on successful call volumes, alarms from many sources, and are usually detected and resolved quickly.

Despite such robust operations support, the system always has a number of call defects occurring at any time of the day in the form of "background noise". Measured in defects per million (DPM), they represent only a small fraction of the calls at any given time, but left unchecked, they can add up quickly over weeks and months. A separate chronics team troubleshoots these defects, but diagnosis is still a largely manual process. We seek to provide tools that can help such chronics teams to quickly discover low-grade problems that are hidden in the background noise.

A. Challenges in Diagnosing Chronics

We illustrate these challenges using examples of real incidents diagnosed using Draco in a production system [10]. Figure 1 illustrates actual instances of chronic problems in the service provider's logs that were discovered using our tool. In the first incident, the recurrent increase in defects during night hours was traced to two different business customers, who were attempting to send faxes overseas using unsupported codecs during US night time. In the second incident, an independent problem with a specific network element arose and persisted until the network element was reset. Figure 2 illustrates a persistent chronic problem due to two blocked CICs (Circuit Identification Codes) on the trunk group that affected calls assigned to these blocked CICs in a round robin manner. At peak, 2–3% of the calls passing this trunk group would fail. After those CICs were unblocked, the total defects associated with this error code were reduced by 80%.

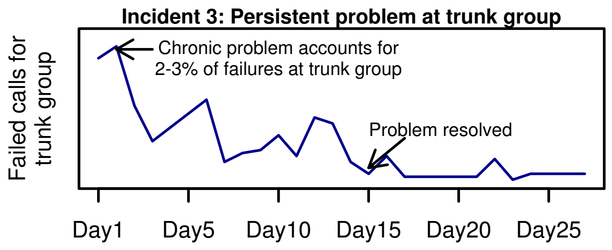


Fig. 2. Chronic problem at production system persists for several weeks making it difficult them to detect using change-points.

These incidents highlight the challenges faced when diagnosing chronics namely:

1) **Chronics fly under the radar.** Chronics occur sporadically, or affect a small subset customers, and thus may not trigger any threshold-based alarms. In the incidents shown in Figures 1 and 2, the defect rate observed by the customers was a fraction of one percent. Setting thresholds to detect these problems is notoriously difficult because lowering the thresholds to detect chronics would increase the number of spurious alarms.
2) **Persistent Problems.** Some problems, occur only for short durations of time, and could be discovered by change-detection algorithms. However, other problems persist for long periods of time as shown in Figure 2. Algorithms that rely on change-point detection methods [1] or those that rely on historical models [9] would fail to detect these problems.
3) **Multiple independent problems.** Because chronics often persist for long periods of time before they are discovered, there are usually many of them ongoing at the same time. Figure 1 illustrates how Draco identified multiple ongoing problems that affected calls passing through the same network element—one related to two different business customers, and one related to the network element
4) **Complex triggers** Chronics often involve only a small subset of user interactions because they are triggered by some unforeseen corner case arising due to a combination of factors. For example, certain chronics arise due to a conflict between the configuration at the customer's premises and the ISP's server. To effectively debug these problems, operators need to know both the server configuration, and the subset of customers affected.

III. Diagnosis Algorithm

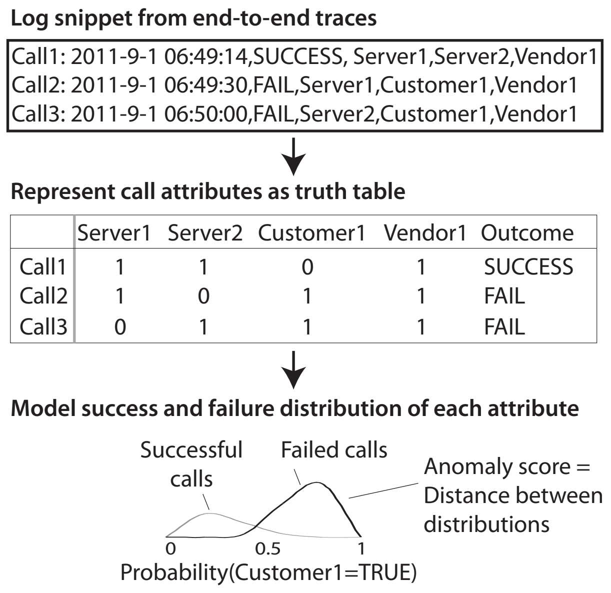
Draco uses a "top-down" approach to localize problems by starting with user-visible symptoms of a problem, *i.e.*, failed calls, and drilling down to identify groups of attributes that are the most highly indicative of the symptoms. We explain Draco's use in the context of a large telecommunication system. However, Draco can be used to diagnose chronics in other large distributed systems, such as e-commerce, web-search, social networks, that have the following characteristics: 1) serve users via independent interactions, 2) log end-to-end traces of user requests, and, 3) label each user interaction as successful or failed.

The diagnosis algorithm proceeds in four steps. First, we label user requests such as phone call attempts as successful or failed, and extract system-level information associated with each call from the server logs. This step is the only domain-specific activity—all other steps are domain-agnostic. Second, we compute an anomaly score for each attribute using a standard information-theoretic metric that represents the "difference" between the success and failure attribute occurrence probability distributions, as shown in Figure 3. Third, we use a scalable ranking function to identify groups of attributes that best discriminate between the success and failure labels, as shown in Figure 4. Fourth, we examine the performance logs of any network elements indicted during the third step, and apply a similar ranking function to identify anomalous performance metrics, such as high CPU or memory usage. We describe each step of the approach in more detail below.

A. Labels and Attribute Extraction

We examined logs from the VoIP network over a period of six months. These include call detail record (CDR) logs that are generated locally by network elements for each call that passes through them. The logs often contain hundreds of attributes that specify details of the call such as the caller and callee information. The structure and semantics of these records are vendor-specific. These logs tend to be large—the average size of the raw CDR logs is 30GB/day. Even after significant consolidation to eliminate irrelevant data fields, the average size is 2.4GB/day, and each log contains between 5000–10000 unique call attributes pertinent to diagnosis, *i.e.*, attributes that appear in defective calls. In addition to CDR logs, we also obtained performance logs collected by the physical hosts on which the network elements run.

We start from user-visible symptoms of a problem, *i.e.*, failed attempts to make a phone call. Labeling of user interactions into success and failure interactions is easy if logs at the user end device are available. However, if only logs from network elements are available as in our case, domain-specific heuristics will often be required. For phone calls, a user redialing the same number immediately after disconnection, zero talk time, or server reported error code can be used as the failure indicator. In other systems, similar heuristics could work too, *e.g.*, a user repeatedly refreshing a web page, or getting a HTTP error when accessing a page. Since these labels are used for subsequent statistical analysis, occasional mislabeling can be tolerated. We then correlate the lower-level system log data extracted from the raw CDRs with these user-level events (phone calls) to construct a "master record" that represents the consolidated end-to-end trace. The log data must have some common keys such as time, phone numbers, and IP addresses that can be used to correlate the data with the user-level event. However, the matches need not be exact, and domain-specific matching rules can be used. For example, entries may belong to the same call if the sender and receiver


Log snippet from end-to-end traces

```
Call1: 2011-9-1 06:49:14,SUCCESS, Server1,Server2,Vendor1
Call2: 2011-9-1 06:49:30,FAIL,Server1,Customer1,Vendor1
Call3: 2011-9-1 06:50:00,FAIL,Server2,Customer1,Vendor1
```

Represent call attributes as truth table

	Server1	Server2	Customer1	Vendor1	Outcome
Call1	1	1	0	1	SUCCESS
Call2	1	0	1	1	FAIL
Call3	0	1	1	1	FAIL

Model success and failure distribution of each attribute
Successful calls
Failed calls
Anomaly score = Distance between distributions
0
0.5
1
Probability(Customer1=TRUE)

Fig. 3. Draco first extracts attributes from the labeled end-to-end traces and represents them as truth table. Next, Draco computes an anomaly score as the distance between the successful and failed distribution for each attribute.

phone numbers match in all available digits and timestamps are within a small window of each other. Besides these keys, the remainder of each log entry is not required to have any special semantic meaning. We can treat it simply as a bag of words. For our VoIP dataset, the end result is a list of "master CDRs", one for each phone call or call attempt, and each labeled as a success or failure as shown in Figure 3.

Each master CDR consists of a number of attributes including the caller and callee phone numbers, call time and duration, network element names and IP addresses used to process the call, any defect and success codes generated by the element, the trunk lines used, and other fields present in the CDRs. Domain knowledge can be used to choose which attributes to include from the original raw logs.

B. Scalable Anomaly Score Computation

Given the set of master records, we then rank attributes using a scoring function that quantifies the ability of the group to discriminate between successful and failed calls. To do so, we use an iterative Bayesian approach to learn a simple Bernoulli ($i.e.$, "coin toss") model of successes and failures. The idea is to model an attribute a as occurring in a call with a fixed, but unknown probability p^a. This attribute occurrence probability is p_f^a for failed calls, and p_s^a for successful calls. The model estimates these unknown probabilities using the master CDRs. However, rather than learning a single value, we can estimate the entire probability *distribution* of these unknown attribute occurrence probabilities, $i.e.$, $F_f^a(x) = P[p_f^a \leq x]$, and $F_s^a(x) = P[p_s^a \leq x]$. We start with an initial estimate for F_f^a and F_s^a, and Bayes rule is used to update this estimate as each new call in the dataset is processed, depending on whether it is a successful and failed call, and whether it contains the attribute a or not. Once these distributions are learned, the score is simply the KL divergence [12], a standard information theoretic metric of the "difference" between two distributions, computed between these success and failure attribute occurrence probability distributions.

We can compute the score for large numbers of attribute groups and over large CDR volumes efficiently because the KL divergence can be reduced to a closed form equation due to two textbook results. The first result is that Beta distributions are *conjugate priors* for Bernoulli models, $i.e.$, if a Beta distribution $Beta(x, y)$ is used as an initial estimate for distribution F_f^a (or F_s^a), and the forward probability $P[a$ appears in a failed call$|F_f^a]$ (and similarly for successful calls) is given by a Bernoulli distribution, then the new estimate for F_f^a after applying Bayes rule is also a Beta distribution $Beta(x+a, y+b)$, where a and b are the number of calls with and without attribute a, respectively. The second result is that the KL divergence between two Beta distributed random variables, $X \sim Beta(a, b)$ and $Y \sim Beta(c, d)$ is given by the Equation

$$\begin{aligned} KL(Y||X) &= \ln\frac{B(a,b)}{B(c,d)} - (a-c)\psi(c) - (b-d)\psi(d) \\ &\quad +(a-c+b-d)\psi(c+d) \qquad (1) \end{aligned}$$

where B is the Beta function and ψ is the digamma function. Therefore, if one starts with the initial assumption that the failure and successful call attribute occurrence probabilities p_f and p_s are uniformly distributed (which is a special case of the Beta distribution), then setting $a/b = 1+$#successful calls with/without attribute a, and $c/d = 1+$#failed calls with/without attribute a in Equation 1 yields the desired score in Equation 1. A similar observation is used to compute KL divergences between two Bernoulli models in [14].

Figure 3 shows how the scoring works in terms of the density functions for the success and failure attribute occurrence probability distributions. Intuitively, it scores higher those attribute groups that are more likely to occur in failed calls than in successful calls, but it does so while taking into account the volume of data observed. This allows us to increase confidence as we observe more calls. For example, the score is higher after observing an attribute in 50 out of 100 failed calls as compared to observing it in 1 out of 2 failed calls, even though both scenarios have the same underlying probability p_f of 0.5.

C. Attribute Group Generation

Chronics can arise due to complex triggers involving a combination of factors such as conflicting software versions on different network elements. These complex conditions are represented as conjunctions of groups of attributes, e.g., "Customer1 and LocationA and CodecB". Draco identifies these groups using a search tree as shown in Figure 4. The root node of the tree represents the null set, and each branch represents a single attribute. Each non-root node of the tree represents a unique attribute group as specified by the path from the root to that node, and the weight of the node is the anomaly score for node's attribute group. Starting with the direct children of

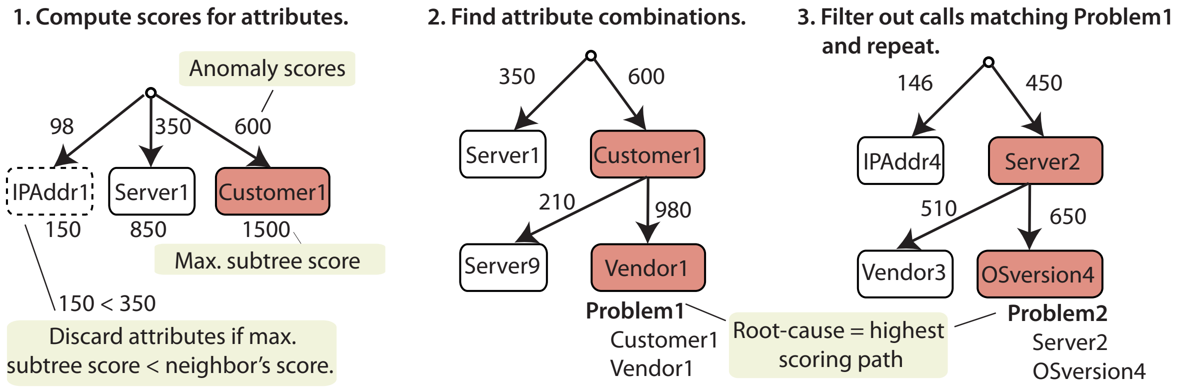


Fig. 4. Draco uses an iterative Bayesian approach to rank combinations of attributes most correlated with the problem.

the root (representing a single attribute each), Draco expands the tree to a depth of d to consider all groups containing up to d attributes. Expanding along a branch of the tree involves a filtering operation that retains only those successful and failure events in which the attribute represented by that branch was present. The filtering is required to get the success and failure counts needed for the anomaly score computation. These filtering operations dominate the algorithm's running time and dictate the data structures used in Draco's design as described in Section IV. The node with the highest weight is picked as the dominant problem signature in that iteration.

For this process to be practical, there are two additional complications that must be handled. The first is to find any attributes that are synonyms of each other. For example, attributes such as a particular customer's IP address and name, or a customer's IP address and a dedicated IPBE server assigned to that customer, may appear together in all calls. Such overlapping attributes are indistinguishable from a statistical point of view but may be meaningful to an operator from a semantic standpoint ($e.g.$, an operator may know how to investigate an IPBE server but not know how to investigate the customer's IP). Therefore, at each node of the tree, we identify all its equivalent attributes and represent the entire set by a single canonical attribute when expanding the tree. The threshold used to mark two, or more, attributes as synonyms is referred to as the *overlap probability* and is a user-configurable parameter that is typically set to a high value such as 0.99. However, when presenting the problem signatures to the operator, we show all the synonyms associated with the attributes identified in the signature.

The second complication is one of scalability. Because tens of thousands of attributes can be present in the dataset, a brute-force approach that expands the entire tree up to depth d is infeasible. To explore attribute groups optimally, Draco uses a branch-and-bound algorithm [13] to dynamically determine the maximum breadth of the tree to explore. Specifically, for each unexplored node of the tree n, Draco computes an upper bound for the maximum anomaly score that can be achieved by any child node n. If this upper bound is lower than the maximum anomaly score seen so far, then exploration of n is guaranteed to be fruitless, and it is discarded without further exploration. The upper bound of the anomaly score for a subtree, $e.g.$, $customer1$ in Figure 4, can be shown to be attained by assuming that there is a branch of that subtree that explains all the failed calls in the subtree, and has zero successes as computed by Equation 2.

$$\begin{aligned} KL^b(Y||X) \quad = \quad & \ln \frac{B(1,a+b-1)}{B(c,d)} - (1-c)\psi(c) - \\ & (a+b-1-d)\psi(d) + \\ & (a+b-c-d)\psi(c+d) \end{aligned} \tag{2}$$

For example, if the attribute $customer1$ was associated with 100 failed calls and 10000 successful calls, then the maximum possible anomaly score for the subtree anchored at $customer1$ would be a branch with 100 failed calls and zero successes.

We iteratively apply this algorithm in a greedy fashion to identify multiple concurrent problems by removing all calls (both success and failures) that match this identified problem signature from the dataset, and repeat the process. Doing so removes the impact of the first diagnosed problem and allows us to ask what explains the remaining failures. In this manner, we can identify separate independent failure causes (see Steps 2 and 3 in Figure 4).

The average complexity of our algorithm is $M * N * D^r$, where M is the number of attributes, N is the number of calls, D is the average depth of the tree, and r is the average degree of nodes in the tree. The magnitudes of D and r are determined dynamically by the Draco's branch-and-bound algorithm.

D. Real-Valued Resource Counters

The Bayesian approach presented in Section III-B is scalable enough to be directly applied to the large numbers of discrete attributes present in our data sets. However, it is difficult to construct such numerically cheap comparison techniques to compare between success and failure distributions of real-valued data. To overcome this limitation, we analyze only a subset of the real-valued data that is linked to attributes that are implicated in signatures produced by the Bayesian analysis.

Specifically, the real-valued data in the VoIP dataset includes performance logs of any network elements within the service

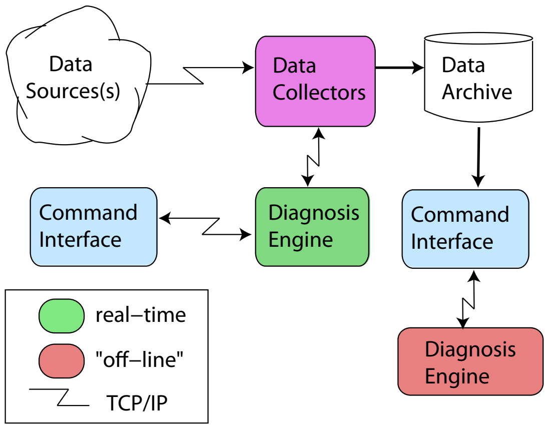


Fig. 5. Draco's flexible architecture supports multiple data sources, and the diagnosis engines can run in either real-time or offline mode.

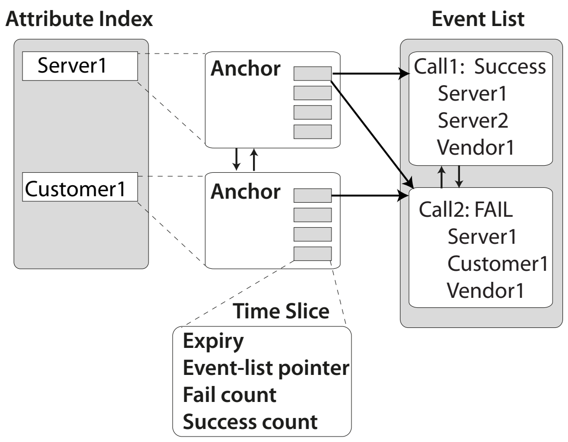


Fig. 6. Draco achieves high performance by maintaining in-memory indices of attribute and event data.

provider's network. These performance logs include periodic measurements (at 5–15 minute intervals) of CPU and memory utilization, network traffic, disk I/O, and other OS-level metrics. For each problem signature identified in Section III-C, Draco considers only those performance measurements associated with network elements present in the signature. We identify the resource-usage metrics that are highly correlated with the problem by annotating each call that matches the problem signature with the resource-usage metrics gathered during the same time interval, as illustrated in the log snippet below.

```
# Resource−usage metrics for server2
# Timestamp,CallNo,Status,Memory(%), CPU(%)
20100901064914,1,SUCCESS,54,6
20100901065530,2,FAIL,82,4
20100901070030,3,FAIL,75,20
```

Next, we use the Wilcoxon rank-sum test [16] to determine whether the distribution of each metric in failed calls differs significantly from the distribution of each metric in successful calls. The Wilcoxon rank-sum test does not assume that the data is drawn from any particular distribution, and assesses whether one of two samples of independent observations tends to have larger values than the other. Comparing the distribution of metrics between successful and failed calls within the same time interval makes Draco more robust to seasonal variations in load ($e.g$, night-time vs. day-time).

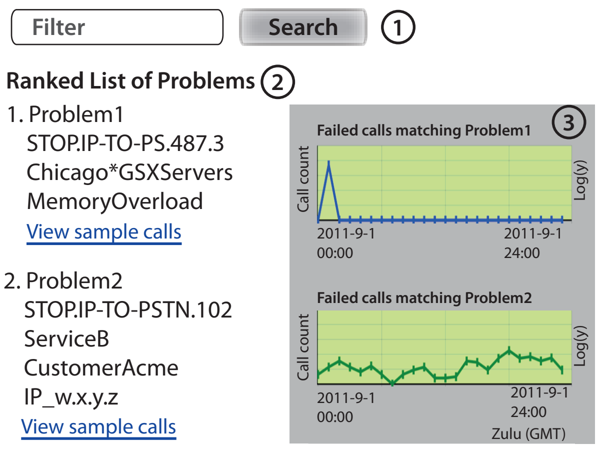
IV. Architecture and Design

We have implemented a prototype of Draco, written in C, which is comprised of data collectors that process consolidated end-to-end call traces to extract attributes of interest, and a diagnosis engine that outputs a ranked list of problems identified (see Figure 5). For the past year, this prototype has been in active daily use by the chronics team at the large ISP to analyze the production VoIP platform. The prototype is flexible since it can be easily extended to incorporate additional sources of information, such as software versions and Quality of Service (QoS) data. In addition, the prototype is scalable and capable of handling tens of millions of calls in real-time even when running on a single server.

The data collectors extract attributes from server logs, and archive the processed logs. Each data collector supports one or more data formats specified using configuration files, which increases the flexibility of our prototype. The data collectors also send data to the diagnosis engine, which implements the algorithms described above in Sections III-B and III-C. The diagnosis engine can receive data from concurrent input sources ($i.e.$, multiple collectors) to reduce the amount of time needed to load data. The diagnosis engine can also be run in an offline mode by reading processed logs from the data archive.

The diagnosis engine considers each end-to-end call record as an event. The engine collects and manages events over a user-controlled time window of length T seconds (the chronics analysis team typically uses a window size of a whole day). Timestamp information in the event data is used to determine the bounds of the window; as new data is received, the window progresses forward and old events are aged off.

Performance was a primary concern while architecting the diagnosis engine as it is necessary to manage thousands of attributes from the VoIP system in real-time. The filtering operations involved in the exploration of the search tree and in filtering out data that can be explained by a newly discovered problem signature, as described in Section III-C, are the most expensive operations of each analysis. This is because each filtering operation must operate on the entire dataset consisting of both successful and failed events, which, despite reductions due to sampling, can still be very large. To construct appropriate data structures for this process, we use the observation that if each event is treated as a "document" that contains words corresponding to each attribute, then computation of the anomaly score for an attribute group involves "searching" both the success and failure document sets for that group of attribute keywords, and counting the matches. Therefore, as shown in Figure 6, Draco's core data structures are constructed similarly to search engines—using an inverted hash table to



1. Problem1
   STOP.IP-TO-PS.487.3
   Chicago*GSXServers
   MemoryOverload
   View sample calls

2. Problem2
   STOP.IP-TO-PSTN.102
   ServiceB
   CustomerAcme
   IP_w.x.y.z
   View sample calls

Fig. 7. Draco's dashboard allows operators to: (1) specify a search criteria such as problems on given date; (2) view a ranked list of chronics diagnosed; and (3), identify recurrent problems using plots of affected calls.

index attributes.

The inverted index maps each attribute back to a linked list of events that contains the attribute. These mappings allow linear time computation of set intersections so that success and failure counts can be quickly constructed for any conjunction of attributes and negated attributes (to support exclusion of events that match previously discovered signatures). For each attribute, a series of success and failure counts are maintained based on the time slices. Managing the counts by time allows them to be adjusted as the time window rolls forward without the need to recount across all unexpired events.

A. Success Event Sampling

Due to the nature of chronics, the datasets Draco processes usually have a disproportionately larger number of successful events as compare to failures. Sampling successful events as they are read by the diagnosis engine yields both a significant reduction in overall memory utilization, and also significantly reduces the time to perform an analysis. To sample, we bin each successful event based on its time slice, and keep 1 out of every Nth successful event in each bin. Unbiased random sampling preserves the correctness of the Bayesian estimation of success and failure distributions as described in Section III-B, and thus preserves the correctness of the anomaly score of Equation 1. Its only impact is to reduce the number of success events, and thus the uncertainty of the success distribution. The results from our production runs of Draco, discussed in Section VI, shows that sampling does not appreciably impact Draco's accuracy, but does increase its speed by more than two orders of magnitude.

B. Visualization

Operators access the prototype via an interactive web-based user interface. Figure 7 illustrates how the web-interface facilitates the operator's workflow.

1) The operator searches for the date and the types of problems they are interested in analyzing. For example, operators can restrict the analysis to calls for a specified VoIP service on a given date.
2) Next, operators are directed to an interactive web-interface interface that ranks the top-20 problems diagnosed by Draco that match their filter, sorted by decreasing severity. Operators can gain more insight on the nature of the problem by viewing samples of calls affected via a drop-down option. The call samples display additional information from the call detail records, such as telephone numbers and call durations, that might not be captured by Draco's problem signature.
3) A plot showing the frequency of the problem is displayed on the right, providing insight on the duration and severity of the problem.

V. Results from Fault Injection Study

We conducted a fault injection study to investigate the effectiveness of Draco under a variety of precisely controlled synthetic faults so that ground truth was known. We also benchmarked our approach against Pinpoint [11] and Spectroscope [20].

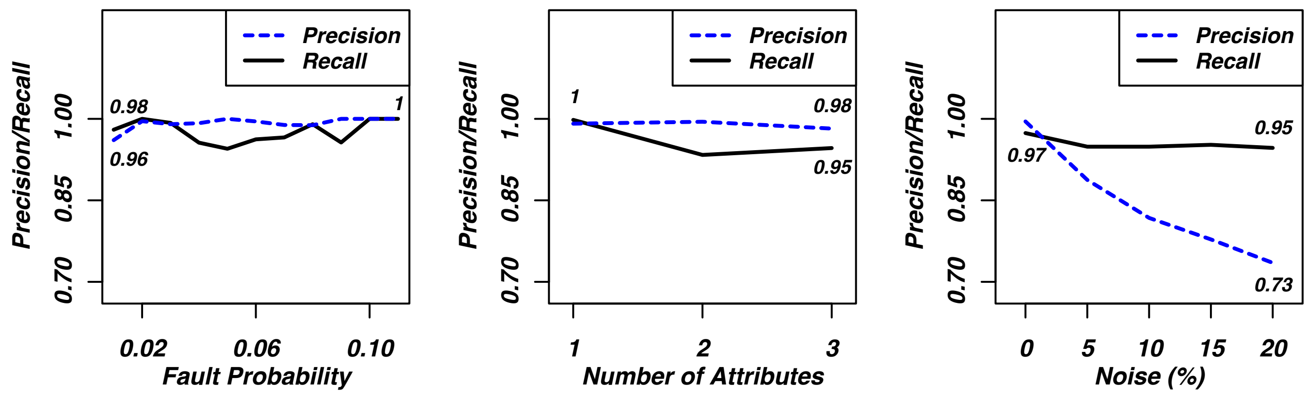
A. Fault Injection Dataset

We simulated faults using actual call-detail records (CDRs) of successful calls from the VoIP production system. We divided the CDRs into 1-hour intervals to yield 500 hourly traces. We injected faults by changing the labels of successful calls, which contained attributes of interest, to failed calls. The attributes of interest were individual network elements, customer sites, links (routes), and their combinations. These attributes were selected because they were the most common features tracked by the operations team at the large ISP.

We randomly varied the combination of attributes associated with each fault from 1 to 3, and ensured that these attributes were not synonyms of each other. We also varied the number of independent faults in each hourly trace from 1 to 3. The probability of each fault injected ranged from 1% to 10% of calls containing the chosen attributes. In addition, we investigated the effect of mislabeled data by incorrectly labeling 5–20% of failed calls as successful, and randomly labeling an equivalent number of successful calls as failed.

We sought to answer the following questions through fault injection: 1) how does varying the fault probability impact the effectiveness of diagnosis? 2) how well can we diagnose complex failures involving multiple attributes? 3) can we identify multiple concurrent faults? and 4) how does noise due to mislabeled data affect diagnosis?

We evaluated the effectiveness of Draco based on the rank of the correct root-cause in the diagnostic output, and computed *recall* and *mean-average-precision*. Recall is the fraction of injected faults that were correctly identified in the top-20 root-causes. Mean-average-precision is a measure of the false positive rate, which is typically used to analyze the quality of ranked search results. A high mean-average-precision indicates that the algorithm had low false positive rates, and ranked the relevant root-causes at the top of the list.



(a) Draco's precision and recall remained relatively constant despite variations in fault probability. (b) Draco correctly identified complex problems involving a combination of attributes. (c) Draco's recall is robust to noise, and its precision is degraded in proportion to noise.

Fig. 8. Effect of variation in fault probability, number of attributes associated with each fault, and noise on Draco's performance.

B. Draco's Fault Injection Results

Draco successfully diagnosed 97% of faults injected, with lower than 4% false positives. All the false negatives occurred when we injected two faults that were logically independent, but happened to share a large intersection of attributes correlated with the faults. In these cases, Draco typically reported a single root-cause that listed the shared attributes. A more detailed breakdown of the results of our fault injection experiments is provided below.

1) *Draco is robust to variations in fault probability.* Figure 8(a) shows that Draco correctly identified the root-cause of injected faults despite variations in the fault probability; Draco's precision and recall remained relatively constant at >96% and >94% respectively.
2) *Draco correctly diagnoses chronics triggered by complex conditions.* Figure 8(b) shows that Draco correctly diagnosed chronics triggered by the interaction of two or more attributes. Draco's precision and recall were slightly degraded from 99% to 98%, and 99% to 93% respectively for chronics involving multiple attributes. As explained above, this drop in recall was due to the presence of faults that were not truly independent rather than the number of attributes associated with each fault.
3) *Draco is effective at diagnosing multiple concurrent faults.* Draco correctly ranked 97% of the relevant root-causes within the top-3 likely causes of chronics. This high ranking of likely root-causes allows operators to quickly focus their attention on the most pressing issues.
4) *Draco tolerates noise due to occasional mislabeling.* Figure 8(c) shows that Draco's recall is robust to noise, and that precision is degraded in proportion to noise. The drop in precision is due spurious attributes introduced by the incorrect labels. Draco's ranking of likely causes remained robust to noise—even when 20% of failed calls were mislabeled, Draco correctly identified >94% of injected faults within the top-3 likely causes.

C. Benchmarking Against Existing Algorithms

We benchmarked our approach against Pinpoint [11] and Spectroscope [20]. These diagnosis algorithms are most similar to Draco as they rely on truth tables, and decision trees that use information-theoretic splitting functions to identify attributes most indicative of failures. We implemented the decision tree algorithms using $See5$ [19], an open-source implementation of the C5.0 algorithm written in C++.

1) Pinpoint: We implemented Pinpoint [11] by training a decision tree using the labeled failed and successful calls. We then diagnosed problems by examining each branch in the decision tree whose leaf node classified failed calls, and ranked the branches based on the number of failed calls. We observed that precision and recall were primarily influenced by the ratio of failed calls to successful calls in the dataset, as shown in Figure 9. We varied this ratio by randomly sampling successful calls, while leaving the number of failed calls unmodified. The best performance was achieved when the ratio of failed to successful calls was similar. Weiss and Provost [21] explain that the performance of decision tree algorithms is degraded when class distributions are imbalanced—these imbalances are commonplace when diagnosing chronics as the number of successes significantly exceeds the number of failures. An example of this degraded performance is shown in Figure 9 where recall dropped to 48% when the number of successful calls outweighed the number of failed calls by a factor of 100. In this case, often the best predictor was a decision tree with no branches that always predicted success.

2) Spectroscope: Spectroscope [20] localizes the source of performance degradations between two periods or executions of a system to just a few relevant components. It does so by leveraging the insight that such changes often manifest as changes or *mutations* in the structure of individual requests ($e.g.$, the components visited, the functions executed, $etc.$) or in their per-component latencies. Spectroscope identifies mutated request flows from the problem period and localizes the problem by showing how they differ from their *precursors*—

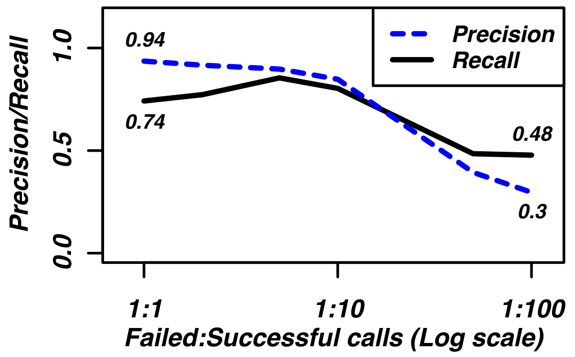


Fig. 9. The performance of decision trees is influenced by the ratio of failed to successful calls in the dataset. Performance degrades significantly when successful calls greatly outnumber failed calls in the dataset.

the way they were serviced in the non-problem period. Additional localization is performed by using a decision tree to identify low-level parameters ($e.g.$, function calls) that best differentiate a mutation from its precursor.

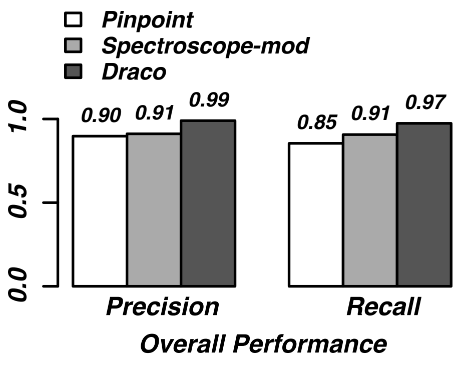
The fault models for Spectroscope and Draco are different—Spectroscope targets problems that result in significant performance degradations, whereas Draco targets chronics. Therefore, we implemented a modified version of Spectroscope-mod where successful calls represent the non-problem period, and failed calls represent the problem period. We investigated whether sampling successful calls using the notion of *precursors* ($i.e$, successful calls that were similar, but not identical to failed calls), yielded better results than the random sampling we employed for Pinpoint. We identified precursors by sampling successful calls whose string-edit distance from failed calls was below a predefined threshold. We then localized the root-cause of the problem using decision trees.

D. Benchmarking results

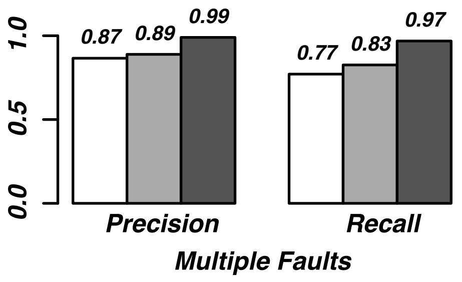
Figure 10(a) summarizes the overall mean-average-precision and recall of Pinpoint[2], Spectroscope-mod, and Draco when diagnosing injected faults, in the absence of noise. Draco performed better than both Pinpoint and Spectroscope-mod by identifying 97% of injected faults with an average precision of 99%. The precision of Pinpoint and Spectroscope-mod were comparable, at 90%. Spectroscope-mod's recall was 6% higher than Pinpoint's demonstrating that strategic sampling of success data can improve performance.

The differences in performance between Draco and the decision tree approaches were more pronounced when we limited our analysis to fault injection traces that either contained multiple independent faults, or chronics triggered by complex corner cases involving a combination of two attributes. Draco correctly diagnosed up to 20% more injected faults for traces containing multiple independent faults, as illustrated in Figure 10(b). Draco significantly outperformed the decision tree approaches for chronics triggered by a combination of two

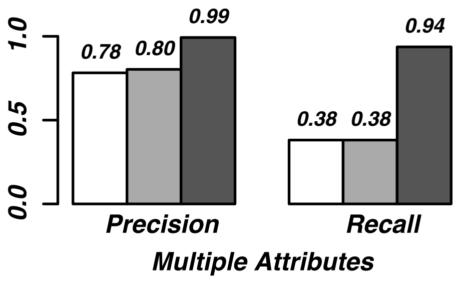
[2]For Pinpoint and Spectroscope-mod, we sampled successful calls to yield a 1:5 ratio of failed to successful calls, which provided the best performance.



(a) Overall, Draco performed better than both Pinpoint and Spectroscope-mod at diagnosing chronics.



(b) Draco's recall was higher by up to 20% for traces containing multiple independent faults; Spectroscope-mod performed 6% better than Pinpoint for these traces.



(c) Draco outperformed both Pinpoint and Spectroscope-mod, with a recall of up to 56% better for complex chronics triggered by a combination of 2 or more attributes.

Fig. 10. Benchmarking Draco against Pinpoint and Spectroscope-mod.

or more attributes, achieving a recall of up to 56% higher as shown in Figure 10(c).

The reasons for the degraded precision and recall for Pinpoint and Spectroscope-mod are outlined below:

1) The decision tree performed poorly at diagnosing faults injected with low probabilities, particularly in traces containing multiple concurrent faults. In these instances, the decision tree algorithm would split the tree to classify faults occurring at higher probabilities, thereby masking faults with lower probabilities.
2) The performance of the decision tree is degraded when chronic problems arise due to a combination of attributes because it identifies some, but not all relevant root-causes. We took great care to ensure that the combination of attributes associated with each injected fault

were not synonyms of each other to eliminate this as contributing factor to the poor performance. We investigated the effect on performance of considering partial matches where at least one of the affected attributes is identified. In this case, the recall of both Pinpoint are Spectroscope-mod improved to 83% suggesting that the decision tree was pruning relevant features.

VI. Draco in Practice

We have deployed Draco on a portion of wireline VoIP services provided by a major ISP. Over the past year, Draco has assisted operators in performing chronics analysis of dropped and blocked calls on the production system. We evaluated the effectiveness of Draco using a diverse set of real incidents from a production telecommunication system, listed in Table I. We ran our experiments on a 8-core Xeon HT (@2.4GHz) with 24GB of memory.

A. Real Incidents

We evaluated Draco against chronics with known root-causes from the production system; see Table I. The root-causes of the chronics included configuration problems at the customer premises, resource contention, software problems, and an intermittent power-outage. Draco correctly localized the network element or customer associated with the chronic problem in 8 out of these 10 incidents. Once the problem is localized by Draco, operators can promptly liase with customers, or query logs outside Draco's scope to diagnose the problem in more detail. The two incidents in which we did not implicate the correct element were a software problem in a policy server (incident 9) and a power outage that resulted in intermittent problems during failover (incident 10). In both incidents, the network element that was the root-cause of the problem was not present in our input data so Draco indicted the network elements adjacent to the root-cause.

In addition to localizing network elements associated with the chronic problem, Draco analyzed the performance logs of the identified network element whenever they were available. Draco flagged a resource metric as anomalous if the distribution of the metric in failed calls was significantly different from that in successful calls. Draco used the Mann-Whitney rank test to reject the null hypothesis that the real-valued metrics associated with failed and successful calls were drawn from the same distribution with a significance-level of 1%. The test helped to localize problems due to resource-contention at a network element.

B. Case Studies

We highlight four case studies from Table I, to illustrate how Draco has been used by the chronics team quickly to identify several new problems.

Incident 4: Poor call quality (due to packet delay, jitter, and packet loss) is a chronic problem that is difficult to detect because the call is neither blocked nor dropped and thus appears as a successful event from the system's point of view. To diagnose poor call quality, the gateway servers were configured to log the message-loss percentage for each call. In addition, the Draco data collector was modified to ignore failed (dropped and blocked) calls and treat the set of calls with poor quality (loss $>$ threshold) as the new set of failed calls. The resulting data can then be analyzed normally by Draco's diagnosis engine.

Draco indicated that the top quality of service issue (approximately 48% of all poor quality calls) was related to a single business customer. Further, Draco did not implicate any network elements indicating that the root-cause of the problem was likely with the customer equipment and not a problem with the ISP's hardware and/or network. When the customer was notified, and the problem corrected, the overall number of quality of service failures was reduced as expected.

Incident 5: A business customer experienced extremely high call volumes which resulted in intermittent congestion on bundles between two gateway servers and a switch. Draco identified the network element associated with the customer (*i.e.*, the customer's IPBE), and determined that the problem was correlated with high concurrent sessions and CPU usage.

Incident 6: An intermittent performance problem with two application servers led to an increase in call defects persisting for several days. This problem affected 0.1% of all calls passing through these application servers. Draco identified both servers affected by the problem. After the operations team failed over traffic to a backup server, the number of defects was reduced by 85%. We analyzed the CPU, memory and network-related metrics on the application servers and observed that these failures occurred during periods of heavy load and high CPU usage.

Incident 9: A chronic problem arose when a policy server in the VoIP network stopped responding to invites from application servers, and affected 0.4% of calls passing through the application server. Since records for the policy server were not present in the master CDRs that we analyzed, Draco implicated the application servers that were sending invites to the policy server. An analysis of the performance logs at the application server indicated that low response rates were an additional symptom of the problem. Although in this instance, Draco did not identify the root-cause, our analysis provided useful clues to operators to help localize the problem. The incorporation of more server logs, such as policy server and router logs, would improve our ability to localize problems.

C. Draco's performance

Depending on the operating mode, Draco takes 2 to 6 minutes to load input data, and from 16 seconds to over 10 minutes to analyze input data comprised of more than 30 million calls; see Table II for details.

In the initial implementation, the tree size was limited in order to achieve acceptable analysis times. Enabling the branch-and-bound algorithm described in Section III-C, while continuing to limit the tree size, resulted in more than a 50% performance improvement in the analysis time. However, the branch-and-bound algorithm alone does not provide enough of a performance gain to allow the restrictions on tree size to be

TABLE I
EXAMPLES OF CHRONICS AT PRODUCTION SYSTEM. DRACO CORRECTLY DIAGNOSED 8 OUT OF 10 INCIDENTS AND RANKED THEM AMONG THE TOP-20 PROBLEMS IDENTIFIED. DRACO ALSO IDENTIFIED ANOMALOUS RESOURCE-USAGE METRICS WHENEVER PERFORMANCE LOGS WERE AVAILABLE.

	Examples of problems	**Type**	**Diagnosed**	**Resource anomalies**
1.	Customers use wrong codec to send faxes abroad.	Configuration	✓	-
2.	Customer problem causes recurrent blocked calls at IPBE.	Configuration	✓	-
3.	Blocked circuit identification codes on trunk group.	Configuration	✓	-
4.	Problem with customer equipment leads to poor QoS.	Configuration	✓	-
5.	Congestion at gateway servers due to high call volumes.	Contention	✓	CPU/Concurrent sessions
6.	Performance problem at application server.	Contention	✓	CPU/Memory
7.	Debug tracing overloads servers during peak traffic.	Contention	✓	CPU
8.	Software problem at control server causes blocked calls	Software bug	✓	-
9.	Policy server not responding to invites from application servers.	Software bug		Low responses at app. server
10.	Power outage and unsuccessful failover causes brief outages.	Power		-

TABLE II
DRACO'S AVERAGE DATA LOAD TIME, AVERAGE NUMBER OF NODES IN A DIAGNOSIS TREE AND MEAN ANALYSIS TIME TO GENERATE THE TOP 20 DIAGNOSES FOR MORE THAN 30 MILLION CALLS.

	Mode		**Load Time**	**Nodes**	**Analysis Time**
Branch & Bound	Sampling	Restricted			
NO	NO	YES	374 ± 29sec	429 ± 208	524 ± 128sec
YES	NO	YES	374 ± 29sec	12 ± 5	128 ± 53sec
YES	NO	NO	374 ± 29sec	36 ± 20	880 ± 124sec
YES	YES	NO	120 ± 7sec	40 ± 30	16 ± 6sec

removed; doing so caused the analysis times to exceed those of the initial implementation. Sampling (at the rate of 1/200 of successful calls), when used in combination with the branch-and-bound algorithm, does allow the restrictions on tree size to be lifted while reducing analysis times to a near interactive level. The reduction of data load time by more than 60% is another benefit to the use of sampling.

The problem signatures generated when sampling have a 97% match rate when compared to those generated when all success data is used. Specifically, the analysis of several days' data yielded 220 problem signatures, but only 214 matching signatures were produced by the analysis using sampled input. Of the six unmatched signatures, all but one were ranked either 19th or 20th (out of 20); the exception was ranked 13th.

VII. RELATED WORK

Over the past decade, there have been significant advances in tools that exploit statistics and machine learning to diagnose problems in distributed systems. This list is by no means exhaustive, but we believe it captures the trends in diagnosis. This section discusses the contributions of these techniques, and their shortcomings at diagnosing chronics.

A. End-to-end Tracing

Some diagnostic tools [3], [5], [11], [20] analyze end-to-end request traces and localize components highly correlated with failed requests using data clustering [3], [20] or decision trees [5], [11]. They detect problems that result in changes in the causal flow of requests [11], [20], performance degradation [20], or error codes [5]. These techniques have typically been used to diagnose infrastructural problems, such as database faults and software bugs (*e.g* infinite loops and exceptions) which lead to a marked perturbation of a subset of requests. In principle, techniques such as decision trees should fare well at diagnosing both major outages and chronics. However, decision trees did not fare well at diagnosing chronics when we applied them to our dataset. The decision tree's bias towards building short trees led to the pruning of relevant features when diagnosing problems due to complex triggers. In addition, the small number of calls affected by chronics coupled with the presence of multiple independent chronics degraded the performance of the decision tree.

B. Signature-based

Signature-based diagnosis tools [4], [6], [7] allow system administrators to identify recurrent problems from a database of known problems. These techniques typically rely on Service-Level Objectives (SLOs) to identify periods of time where the system was behaving abnormally, and apply machine learning algorithms to determine which resource-usage metrics are most correlated with the anomalous periods. These techniques can diagnose problems due to complex triggers by localizing the problem to a small set of metrics. However, they do not address multiple independent problems as they assume that a single problem occurs at a given instance of time. Chronic conditions might also go undetected by the SLOs because they are not severe enough to violate the thresholds.

C. Graph-theoretic

Graph-theoretic techiques analyze communication patterns across processes to track the probability that errors [2], [9], or successes (*e.g.*, probes [18]) propagate through the system. Sherlock [2] builds models of node behavior, and diagnoses problems by computing the probability that errors propagate from a set of possible root-cause nodes. NetMedic [9] uses a statistical approach to diagnose propagating problems in enterprise systems. These techniques can detect multiple independent problems—ranking them by likelihood of occurrence. However, they do not address problems due to complex triggers as they assume that the root-cause of the problem stems from a single component. In addition, since chronics do not severely perturb system performance they can be included in the profiles of normal behavior learned from historical data—causing chronics to go undetected.

D. Event correlation

Event correlation has been used to discover causal relationships between alarms across components in supercomputers [17], IPTV networks [15], and enterprise networks [22]. These techniques support diagnosis of multiple independent problems, and might be applicable in our system when there are resource-contention problems due to overload within the service provider's network. However, most of the chronics we have observed are due to customer-site problem such as misconfigurations, and operators at the ISP lack access to customer-site data other than names of the customer—therefore event-correlation might not be possible. Draco localizes these chronics by analyzing data that is causally-related with each call rather than alarm signals across the network.

VIII. Conclusion

This work introduces *chronics*—small problems in large distributed systems that significantly degrade user experience because they persist undiagnosed for lengthy periods of time—and describes Draco, a diagnosis engine that identifies and localizes them. We showed through examples of real chronics in the VoIP platform of a major ISP why they are notoriously difficult to diagnose: their small size makes setting alarm thresholds tricky, there are many of them active concurrently even when the system as a whole is mostly functional, their symptoms often overlap with each other, they are triggered by complex corner cases involving multiple conditions, and they they persist for lengthy periods and can get absorbed into the system's definition of what is normal.

Draco addresses these issues through a variety of techniques: a) using top-down diagnosis starting with abnormal user interactions to identify failures rather than relying on bottom-up alarms based on server logs, b) statistically identifying root-causes by comparing bad interactions with good ones *from the same interval of time* rather than relying on thresholds or on historical data from good intervals of time, c) a branch-and-bound procedure to identify complex triggers comprised of conjunctions of multiple attributes, and d) greedy filtering of failures explained by already identified problems to discover additional concurrent problems.

Draco has been deployed on a major VoIP platform serving millions of users and handling tens of millions of calls a day, and is being successfully used by its operations team. We provide examples of real chronics that Draco has helped identify, and through injection of synthetic failures on a dataset obtained from the production system, have shown that for datasets with tens of million of calls, it can provide coverage levels as high as 97% with false positives as low as 4%, and can do so while providing near-interactive performance of < 1 second per chronic all while running on a single server machine with middle of the range hardware.

References

[1] M. K. Agarwal, M. Gupta, V. Mann, N. Sachindran, N. Anerousis, and L. B. Mummert. Problem determination in enterprise middleware systems using change point correlation of time series data. In *NOMS*, pages 471–482, Vancouver, Canada, April 2006.

[2] P. Bahl, R. Chandra, A. G. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *SIGCOMM*, pages 13–24, Kyoto, Japan, August 2007.

[3] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *OSDI*, pages 259–272, San Francisco, CA, December 2004.

[4] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen. Fingerprinting the datacenter: automated classification of performance crises. In *EuroSys*, pages 111–124, Paris, France, April 2010.

[5] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *ICAC*, pages 36–43, New York, NY, May 2004.

[6] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *SOSP*, pages 105–118, Brighton, United Kingdom, October 2005.

[7] S. Duan and S. Babu. Guided problem diagnosis through active learning. In *ICAC*, pages 45–54, Chicago, IL, June 2008.

[8] FCC. The Proposed Extension of Part 4 of the Commissions Rules Regarding Outage Reporting To Interconnected Voice Over Internet Protocol Service Providers and Broadband Internet Service Providers. Technical Report PS Docket No. 11-82, FCC 12-22, Federal Communications Commission, February 2012.

[9] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. In *SIGCOMM*, pages 243–254, Barcelona, Spain, August 2009.

[10] S. Kavulya, K. R. Joshi, M. A. Hiltunen, S. Daniels, R. Gandhi, and P. Narasimhan. Practical experiences with chronics discovery in large telecommunications systems. *Operating Systems Review*, 45(3):23–30, 2011.

[11] E. Kiciman and A. Fox. Detecting application-level failures in component-based internet services. *IEEE Trans. on Neural Networks: Special Issue on Adaptive Learning Systems in Communication Networks*, 16(5):1027– 1041, September 2005.

[12] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22:79–86, March 1951.

[13] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[14] C. Liu, Z. Lian, and J. Han. How Bayesians debug. In *ICDM*, pages 382–393, Hong Kong, China, December 2006.

[15] A. A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao. Towards automated performance diagnosis in a large IPTV network. In *SIGCOMM*, pages 231–242, Barcelona, Spain, August 2009.

[16] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[17] A. J. Oliner, A. V. Kulkarni, and A. Aiken. Using correlated surprise to infer shared influence. In *DSN*, pages 191–200, Chicago, IL, July 2010.

[18] I. Rish, M. Brodie, N. Odintsova, S. Ma, and G. Grabarnik. Real-time problem determination in distributed systems using active probing. In *NOMS*, pages 133–146, Seoul, South Korea, April 2004.

[19] RuleQuest Research Data Mining Tools. See5/C5.0, 2011. http://www.rulequest.com/.

[20] R. R. Sambasivan, A. X. Zheng, M. D. Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger. Diagnosing performance changes by comparing request flows. In *NSDI*, pages 43–56, Boston, MA, March 2011.

[21] G. M. Weiss and F. J. Provost. Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research (JAIR)*, 19:315–354, 2003.

[22] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High speed and robust event correlation. *Communications Magazine, IEEE*, 34(5):82–90, May 1996.