Automated Diagnosis of System Failures with Fa

Songyun Duan, Shivnath Babu
Duke University

I. Introduction

Recent studies show that popular services routinely suffer user-visible problems like slow responses, blank pages or error messages being displayed, items not being added to shopping carts, unexpected database slowdowns, and others [1]. Walmart.com was unavailable for 10 hours during the peak U.S. 2006 holiday season. Other services like Amazon.com, PayPal, Microsoft.com, U.S. Airways, and Wall Street Journal have faced similar problems [1]. Such problems cause systems to violate *service-level objectives (SLOs)* that specify acceptable levels of service. For example, an SLO for an online brokerage may require all transactions to complete within 1 second.

When a system meets all specified SLOs, it is in a healthy state. SLO violations indicate *failures* that can cause huge losses. Brokerages and banking firms can lose up to $75,000 per minute of downtime. A 22-hour outage at eBay cost the company more than $3 Million in customer credits and $4 Billion in market capitalization. The true cost of system failure is much higher: lost or dissatisfied customers, damage to the company's reputation, impact on the company's stock price, and lost employee productivity [1].

While quick failure diagnosis and system recovery is critical, database and system administrators continue to struggle with this problem. The spectrum of possible causes of failure is huge: performance problems like resource contention, crashes due to hardware faults or software bugs, misconfiguration by system operators, and many others. The scale, complexity, and dynamics of modern systems make it laborious and time-consuming to track down the cause of failures manually.

Conventional data-mining techniques like clustering and classification have a lot to offer to the hard problem of failure diagnosis. These techniques can be applied to the wealth of monitoring data that operational systems collect. However, some novel challenges need to be solved before these techniques can deliver an automated, efficient, and reasonably-accurate tool for diagnosing failures using monitoring data; a tool that is easy and intuitive to use:

- **Noisy data:** Monitoring data collected from production systems contains various types of errors that can mislead diagnosis: (i) natural system variability injects Gaussian noise; (ii) failures may corrupt observations; and (iii) rapid system state transitions cause observations from different states to get mixed up.
- **High dimensionality:** Some of our monitoring datasets have 100-300 attributes per server, posing challenges from an accuracy as well as running-time perspective.
- **Dynamic systems:** Conventional approaches like defining a baseline system behavior, and pinpointing deviations from the baseline do not work when workloads and system configuration change over time.
- **Reuse:** Since failure diagnosis is expensive in large-scale and complex systems, it is valuable to leverage past diagnosis efforts whenever possible; particularly since 50-90% of failures seen are recurrences of previous failures.
- **Trust:** Features like reliable confidence estimates and evidence for diagnosis results are important for an automated diagnosis tool to gain administrators' trust; otherwise the tool will not be used in practice.

Fa[1] is a new system for automated diagnosis of system failures that is designed to address the above challenges. Section II describes the technical contributions of Fa, and Sections III-IV discuss our demonstration plan. We will show Fa in action using real-life failures injected in an eBay-like auction service, and also convey the insights that we have gained from 2+ years of work on automating failure diagnosis using system monitoring data.

II. The Fa System and Contributions

System Monitoring Data: When a system is running, Fa collects monitoring data periodically and stores it in a database. For example, Fa uses the *sar* utility to collect more than 100 performance metrics (e.g., average CPU utilization, number of disk I/Os) periodically from Linux servers. Database servers maintain performance counters (e.g., number of index updates, number of full table scans) that Fa reads periodically. Most enterprise monitoring systems like HP OpenView and IBM Tivoli Monitoring collect similar data. Fa collects 100-300 such metrics periodically from the systems that it monitors.

Over a period of time, the monitoring data collected by Fa will contain three types of instances:

- **Healthy data** H is monitoring data collected when the system was in a healthy state. Recall that a system is in a healthy state when it experiences no SLO violations; and in a failure state otherwise.
- **Annotated failure data** L is monitoring data collected from failure states of the system where the cause of failure has been diagnosed. A successful diagnosis can happen any time after the failure occurs. Upon diagnosis, information about the cause of failure is attached as an *annotation* (or metadata) to the corresponding monitoring data.
- **Unannotated failure data** U is monitoring data collected from failure states of the system where the cause of failure has not been diagnosed so far.

[1] African god of fate and destiny

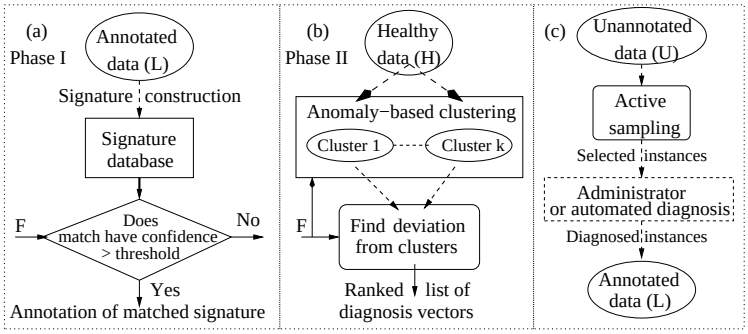

Fig. 1. Control and data flow in Fa: Phase I, Phase II, and the Background Phase

SD_2

s_1	s_2	s_3	s_4	annotation
1	0	0	0	A_1
0	1	0	0	A_2
0	0	1	0	A_3
0	0	0	1	A_4

s_1 = 1 if y > 45, otherwise 0
s_2 = 1 if x < 39, otherwise 0
s_3 = 1 if x > 80, otherwise 0
s_4 = 1 if y < 20, otherwise 0

SD_3

s_1	s_2	s_3	s_4	s_5	s_6	annotation
1	0	0	0	1	0	A_1
0	1	0	0	0	0	A_2
0	0	1	0	0	1	A_3
0	0	0	1	1	1	A_4

s_5 = 1 if 0.36*x + y < 61, otherwise 0
s_6 = 1 if 0.81*x − y < −16, otherwise 0

Fig. 2. Signature databases: (a) SD_2, (b)SD_3

When the monitored system experiences a failure, an administrator or system-management software can diagnose the cause of the failure by posing a *diagnosis query* to Fa of the form *Q=Diagnose*$(F, H \cup L \cup U)$:

- F is monitoring data from the system during the failure (or just before the failure in the case of a system crash).
- $H \cup L \cup U$ is the historic data collected so far.

Fa processes a diagnosis query *Q=Diagnose* $(F, H \cup L \cup U)$ in multiple phases, as illustrated in Figure 1.

A. Phase I: Diagnose(F, L) (Figure 1(a))

Fa first finds whether the failure represented by F is the same as a previously-diagnosed failure in L. That is, Phase I translates the diagnosis query to *Diagnose*(F, L). Fa generates a *signature database* from L that contains entries of the form $\langle sig, A\rangle$ where sig is a signature for the failure with annotation A. Instances F from an undiagnosed failure can be matched against this database. If the diagnosis result produced by this phase has confidence higher than a specified threshold, then Fa returns this result; otherwise Fa goes to the more expensive Phase II of diagnosis.

Because of space constraints, we illustrate our ideas using examples. Suppose L is as shown in Figure 3(a). Each instance has two attributes, x and y, and one of four distinct annotations A_1 (cross), A_2 (plus), A_3 (triangle), or A_4 (circle).

Clustering: One way to generate the signature database is by clustering the data in L using a technique like K-means. Figure 3(a) shows the clusters per failure type. The centroids of these clusters—represented by blue stars ("$\star$") in Figure 3(a)—become the signatures for the corresponding failure type, giving a signature database SD_1.

Suppose $f_1 = \langle 32, 41\rangle$ in Figure 3(a) is a failure instance that we want to diagnose. f_1 can be matched with SD_1 to find the centroid (signature) nearest to f_1; which happens to be a centroid for Failure A_1 (cross) given the data distribution. However, this diagnosis is incorrect since it is obvious from Figure 3(a) that f_1 is an instance of Failure A_2 (plus).

Separating functions: Suppose we identify *separating functions* $s_1(x, y)$ to $s_4(x, y)$ that separate each type of failure instances from the others. Figure 3(a) shows s_1–s_4 as separating lines in the 2D plane. (There are many other forms; Fa uses classification trees.) For example, $s_1(x, y)$ separates the instances of Failure A_1 from the others, and has the form: $s_1(x, y)$=1 if $y > 45$, otherwise 0.

Matrix: Figure 2(a) shows a signature database SD_2 generated using s_1–s_4. SD_2 is a matrix with each row representing the signature of some failure. For example, the signature of Failure A_1 is $\langle s_1(x, y) = 1, s_2(x, y) = 0, s_3(x, y) = 0, s_4(x, y) = 0\rangle$, denoted $\langle 1, 0, 0, 0\rangle$. Each column represents a separating function. For example, the first column represents $s_1(x, y)$ which maps instances of Failure A_1 to 1, and instances of all other failures to 0.

To match instance $f = \langle x, y\rangle$ with SD_2, we compute $\vec{s}(x, y)$ = $\langle s_1(x, y), s_2(x, y), s_3(x, y), s_4(x, y)\rangle$ and find the signature nearest to $\vec{s}(x, y)$ in SD_2. We will measure distances in terms of the *Hamming distance*, namely, the number of bits that are different. For example, $\vec{s}(32, 41)$ is $\langle 0, 1, 0, 0\rangle$ for $f_1 = \langle 32, 41\rangle$. The distances of $\vec{s}(32, 41)$ to the four signatures in Figure 2(a) are respectively 2, 0, 2, 2. Since $\vec{s}(32, 41)$ is nearest to A_2's signature, f_1 is diagnosed correctly.

Handling errors: Now consider f_2=$\langle 39, 41\rangle$ in Figure 3(b). f_2 is of failure type A_2, but has higher error in the x dimension than the instances in L. If we match f_2 against SD_2, $\vec{s}(39, 41)$ is $\langle 0, 0, 0, 0\rangle$. Since $\vec{s}(39, 41)$ is equidistant from all signatures in SD_2, f_2 will not be diagnosed correctly by SD_2.

Now suppose we use the signature database SD_3 from Figure 2(b) to diagnose f_2. SD_3 contains two new separating functions, $s_5(x, y)$ and $s_6(x, y)$, whose separating planes are shown in Figure 3(b). Now, $\vec{s}(x, y)$ = $\langle s_1(x, y), \cdots, s_6(x, y)\rangle$. For f_2, $\vec{s}(39, 41)$ = $\langle 0, 0, 0, 0, 1, 0\rangle$. $\vec{s}(39, 41)$ has least Hamming distance to A_2's signature in Figure 2(b), so f_2 will now be diagnosed correctly.

Why did SD_3 diagnose f_2 correctly, while SD_2 did not? The reason can be understood from an analogy to *error correction* in telecommunications. For f_2, s_2 predicts 0 instead of the correct 1; causing a 1-bit error. SD_3 is robust to 1-bit errors, but SD_2 is not. The Hamming distance between any two signatures in SD_3 is $\geq$ 3. Thus, even though $\vec{s}(39, 41)$ was computed as $\langle 0, 0, 0, 0, 1, 0\rangle$—a 1-bit error from the ideal $\langle 0, 1, 0, 0, 1, 0\rangle$—$\vec{s}(39, 41)$ still remained nearest to A_2's signature.

This example shows that selected redundancy in the set of separating functions can overcome incorrect predictions by some of the functions. In addition, s_2 and s_3 are two less reliable functions in our example because they rely on the noisy x attribute. We can drop s_2 and s_3, and generate a new signature database SD_4 (Figure 3(c)) that is robust to such errors. In summary, Fa's Phase I makes some novel contributions (described further in [2]):

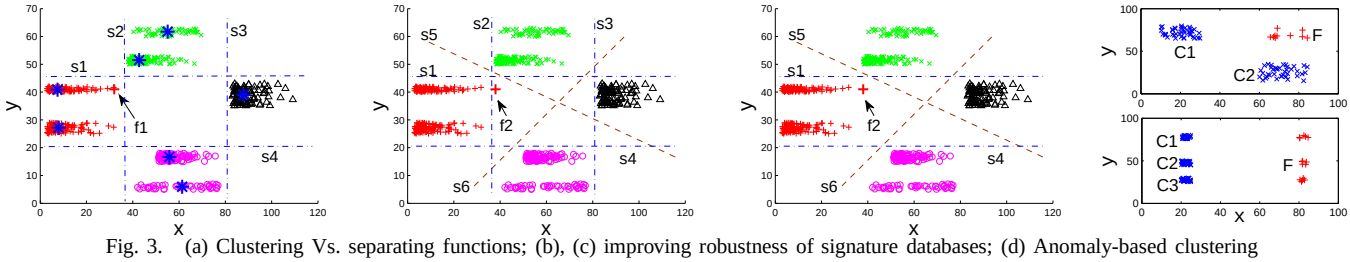

Fig. 3. (a) Clustering Vs. separating functions; (b), (c) improving robustness of signature databases; (d) Anomaly-based clustering

- A powerful representation of the signature database to achieve both good accuracy and robustness to errors.
- Techniques to construct this database from raw system-monitoring data with numeric and categorical attributes.
- Providing confidence estimates with diagnosis results.

B. Phase II: Diagnose(F, H) (Figure 1(b))

The goal of this phase is to determine how the failure instances F to be diagnosed differ from the instances H representing the system in healthy states. We will illustrate the main ideas using examples. Suppose H consists of the instances in Figure 3(d) (top) shown using the "x" symbol. Each instance has two attributes x and y. The figure also shows the failure instances F, indicated using the "+" symbol.

It is clear from the figure that there are two distinct healthy states of the system: C_1 with $x \in [10, 30]$, $y \in [65, 80]$, differing from F primarily along the x attribute; and C_2 with $x \in [60, 80]$, $y \in [15, 30]$, differing from F primarily along the y attribute. A conventional clustering algorithm like K-means or *Locally Adaptive Clustering (LAC)* [3] can identify these clusters in H, and link both attributes x and y to the failure.

Next, suppose H ("x") and F ("+") are as shown in Figure Figure 3(d) (bottom). A conventional clustering algorithm will now group the instances in H into three distinct clusters (C_1, C_2, and C_3 shown in the figure. Since each of these clusters differs from F along both the x and y attributes, both attributes will be linked to this failure as well. However, a closer look at Figure 3(d) (bottom) indicates that this answer is incorrect. Both the failure data and healthy data have similar distribution along the y axis, and differ along the x axis only. So, the correct answer should link the failure to x only.

What went wrong in the second example? Conventional clustering algorithms ignore the failure instances F while deciding how to group the instances in H into clusters. Thus, the clusters generated by these algorithms are independent of the failure instances to be diagnosed, causing two major weaknesses: (i) generating clusters that do not give the correct diagnosis (as in our example), and (ii) generating many more clusters than needed, which can mislead system administrators.

We have developed a new clustering methodology, called *anomaly-based clustering*, where H is clustered with consideration of the instances F to be diagnosed. (That is, the same H may be clustered differently for a different F.) Intuitively, anomaly-based clustering will place two instances $h_1, h_2 \in H$ into the same cluster iff they have similar deviations from F. This strategy gives the right answer for the example in Figure 3(d) (bottom), generating a single cluster for H, and linking the failure to attribute x only.

In summary, Fa's Phase II makes two novel contributions (described further in [2]):

- By giving special attention to F while clustering H, anomaly-based clustering can identify a minimal set of healthy system states needed for accurate diagnosis of F (i.e., no false positives or negatives). It outperforms classic (e.g., K-means) and recent clustering algorithms (e.g., *LAC* [3]) on both efficiency and diagnosis accuracy.
- In spite of the high-dimensionality of monitoring data, Fa produces concise attribute sets while characterizing the deviation of F from healthy states.

C. Background Phase III: $U \rightarrow L$ (Figure 1(c))

Note that Phase II is a general-purpose diagnosis technique, while Phase I can only diagnose new failures that have happened in the past and whose causes have been diagnosed. Nevertheless, Phase II has two drawbacks. First, the "suspicious attributes" in the monitoring data whose distribution differs between H and F may not always identify the *root cause* of the failure unambiguously. For example, a database failure may be attributed to increased lock acquisition times observed in F, but the root cause may be frequent update statements from a new application. Hence, Phase I may need to be supplemented with some (expensive) manual effort to find the root cause or a fix for the failure.

Second, Phase II is a fundamentally harder task than Phase I. In machine-learning terminology, Phase I is a *supervised learning* task and Phase II is an *unsupervised learning* task. Consequently, diagnosis results from Phase II tend to be less accurate than those from Phase I, especially on high-dimensional and noisy monitoring datasets.

These observations about Phases I and II motivated us to consider a new Phase III in Fa where we move data proactively from U to L; with the goal of increasing the accuracy and coverage of Phase I with the least manual effort. Figure 1(c) illustrates this step. Phase III is implemented using a new algorithm, called *Falcon* [4], that can select some instances u from U, and pose an *annotation query* to the administrator of the form: What are the annotations for the instances in u?

To answer this annotation query, the administrator will have to actually diagnose the cause of the failure represented by u. She can take the help of Fa's Phase II for this purpose. If the administrator is able to respond back with the annotations for u, then Falcon will remove u from U, and add u and its annotations to L. Otherwise, these instances are left in U. Falcon then iterates by selecting a new set of instances from U, and posing a new annotation query to the administrator. As new annotated instances appear in L, the signature database

Name	a	i	Description of data and failures
1. Dolphin, 2. ECE	43	4881	OS-level data collected for 55 days from two heavily-used departmental servers at Duke
3. Rubis-bug	110	900	Data access by the *BuyNow* EJB gets null result occasionally (bug in application logic)
4. Rubis-jndi	110	1500	JNDI naming-directory entry of the *SB_SearchItemsByRegion* EJB gets corrupted
5. OLTP-single	42	3660	Occasional CPU contention caused by an application on OLTP server (no disk contention)
6. OLTP-multi	28	696	Both CPU and disk contention caused separately by an application on the OLTP server
7. Rubis-60	110	8184	Contains annotated data about 60 distinct single-EJB failures injected in our testbed
8. Rubis-complex	110	1797	Contains annotated data about 14 distinct multiple-EJB failures injected in our testbed
9. Synthetic	16	10992	Synthetic annotated data about 10 distinct failures; patterns in the data are complex

TABLE I
MONITORING DATA AND FAILURES. COLUMNS a AND i ARE THE NUMBER OF ATTRIBUTES AND INSTANCES RESPECTIVELY

used by Phase I can be retrained on the new L to potentially improve its accuracy and coverage efficiently.

III. Demonstration Session I: Fa in Action

This session will target a general audience interested in seeing an automated diagnosis tool in action. Fa has been evaluated extensively on monitoring data and failures from a production setting, a variety of failures injected in a testbed, and synthetic data; summarized in Table I. Our demonstration will be based on a testbed that runs *Rubis*—a multitier auction service modeled after eBay—on a JBoss application server and a MySQL DBMS. We will inject common failures in each tier of Rubis using a comprehensive failure-injection tool [5] that we have borrowed and extended.

- Software aging—progressive degradation in performance caused by, e.g., memory leaks, unreleased file locks, and fragmented storage space—is a common cause of failure.
- Software faults are a common cause of failure in the application-server tier. We can inject 3 individual causes of failure—software bugs, data corruption, and uncaught Java exceptions—into any of the 25 Java modules (*enterprise Java beans (EJBs)*) that comprise Rubis.
- Contention for CPU, memory, and disk resources is a common cause of failure in each tier.

We can also inject combinations of these individual failure types. All the novel features of Fa will be shown in action by selectively injecting failures:

- We will show the generation and use of Fa's signature database by injecting recurrent failures.
- Injecting previously-unseen and previously-undiagnosed failures will cause Fa's signature database to give low-confidence matches, forcing the invocation of Phase II. Anomaly-based clustering will be illustrated using a cluster visualization tool (*Vista* [6]).
- Cluster visualization will also be used to show how Fa's Phase III picks annotation queries.

IV. Demonstration Session II: Insights on Automated Diagnosis

This session offers researchers and developers interested in the problem of failure diagnosis an opportunity to tap into the insights that we have gained from our work. We will demonstrate these insights by contrasting the diagnosis results and efficiency of different approaches on injected failures.

Whither probabilistic models? An early version of Fa [7] used *probabilistic models* (e.g., Bayesian networks) for diagnosis. These models need too many training samples and learning time on our high-dimensional monitoring datasets. Real-life failure data is very sparse since systems are mostly in healthy states. We discontinued the use of probabilistic models in favor of Fa's current design (simpler models, redundancy, weighting, anomaly-based clustering).

New twists on old problems: Fa has constantly required us to revisit solutions to conventional data-mining problems from a different viewpoint. For example, Fa's signature database generation is a multi-class classification problem. However, with dynamic systems, high-dimensional monitoring data, and sparse failure data, there is high chance that the errors in the failure data F are not present in the historic data; such errors confuse conventional mining algorithms because these errors are in "test data" but not in the "training data".

History is a double-edged sword: Oracle's recent ADDM tool [8] shows the growing interest among database vendors in automated diagnosis of failures. An important way in which Fa differs from ADDM is the use of historic monitoring data in a principled and automated fashion (e.g., reuse of past diagnosis results and comparing F with distinct historical healthy states). If not used carefully, historic data can actually hurt diagnosis (e.g., producing many false positives) rather than help.

Gaining administrator trust: Administrators as a rule are very reluctant to trust a tool if they do not understand how it works. We have taken steps in Fa to give administrators insight into how specific diagnosis results were generated. For example, along with each result from Phase II, Fa provides a subset of the historic data as evidence for the result.

REFERENCES

[1] S. Pertet and P. Narasimhan, "Causes of failure in web applications," Carnegie Mellon University Parallel Data Lab, Tech. Rep. CMU-PDL-05-109, Dec. 2005.

[2] S. Duan and S. Babu, "Automated Diagnosis of System Failures with Fa," Tech. Rep., Mar. 2008, available at http://www.cs.duke.edu/~shivnath/fa/diagnosis.pdf (ICDE 2008 poster).

[3] C. Domeniconi *et al.*, "Locally adaptive metrics for clustering high dimensional data," *Data Mining and Knowledge Discovery*, vol. 14, no. 1, 2007.

[4] S. Duan and S. Babu, "Guided problem diagnosis through active learning," in *Intl. Conf. on Autonomic Computing*, June 2008.

[5] G. Candea *et al.*, "Automatic failure-path inference: A generic introspection technique for internet applications," in *Proc. of IEEE Workshop on Internet Applications*, 2003.

[6] "Visual Cluster Rendering System," www.cc.gatech.edu/projects/disl/VISTA.

[7] S. Duan and S. Babu, "Proactive identification of performance problems," in *ACM SIGMOD*, June 2006, (Demonstration).

[8] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood, "Automatic performance diagnosis and tuning in Oracle," in *CIDR*, 2005.