Fa: A System for Automating Failure Diagnosis

Songyun Duan, Shivnath Babu, Kamesh Munagala
Department of Computer Science, Duke University

***Abstract*—**

Failures of Internet services and enterprise systems lead to user dissatisfaction and considerable loss of revenue. Since manual diagnosis is often laborious and slow, there is considerable interest in tools that can diagnose the cause of failures quickly and automatically from system-monitoring data. This paper identifies two key data-mining problems arising in a platform for automated diagnosis called *Fa*. Fa uses monitoring data to construct a database of *failure signatures* against which data from undiagnosed failures can be matched. Two novel challenges we address are to make signatures robust to the noisy monitoring data in production systems, and to generate reliable confidence estimates for matches. Fa uses a new technique called *anomaly-based clustering* when the signature database has no high-confidence match for an undiagnosed failure. This technique clusters monitoring data based on how it differs from the failure data, and pinpoints attributes linked to the failure. We show the effectiveness of Fa through a comprehensive experimental evaluation based on failures from a production setting, a variety of failures injected in a testbed, and synthetic data.

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 [18]. 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 [18]. 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 [18].

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 [8].

We show in this paper that 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.

Let D denote the (historic) monitoring data collected from a system over a period of time. Let F denote the monitoring data from the same system during a recent failure (or just before the failure in the case of a system crash). A common approach in diagnosis is to use D to build a *baseline* of the system behavior, and to characterize how F differs from this baseline (i.e., how is F anomalous?). Unfortunately, this approach fails in dynamic systems—e.g., where workloads or configurations change over time—which can have multiple distinct states with very different (baseline) behavior. This hurdle can be overcome using a clustering algorithm to partition D into clusters that each represent a distinct baseline.

The conventional approach to clustering (e.g., K-means or subspace-clustering algorithms like *locally adaptive clustering (LAC)* [11]) uses metrics that measure the similarity among instances in D. Diagnosis results are then computed based on how F differs from the clusters found. Unfortunately, this approach can generate (i) clusters that lead to incorrect diagnosis, or (ii) many useless clusters that confuse administrators.

We address this problem using a new clustering metric that considers how instances in D differ from the specific failure instances F to be diagnosed. This new twist on clustering, called *anomaly-based clustering*, ensures that the right clusters to diagnose F are generated. This approach leads to accurate diagnosis results, but it is nontrivial to develop efficient algorithms. We propose novel algorithms for anomaly-based clustering that blend accuracy with efficiency, and can pinpoint causative attributes for real failures in production systems.

While the baselining-based approach to diagnosis is useful and general-purpose, it has two drawbacks. First, the "suspicious attributes" in the monitoring data whose distribution differs between D 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, baselining may need to be supplemented with some (expensive) manual effort to find the root cause or a fix for the failure.

Second, [5] reports that typically 50%, and sometimes as much as 90%, of all software problems reported by users today

	time	lock_time	num_io	failures
h1	1	51.3	76.1	0
h2	2	48.5	63.6	0
h3	4	51.9	97.9	0
h4	5	49.0	43.6	0
h5	8	50.4	13.5	0
h6	9	50.8	51.2	0
h7	11	49.8	119.5	0
h8	12	49.3	141.2	0
h9	13	72.4	65.4	0

(a) Healthy data H

time	lock_time	num_io	failures	annotation
3	95.4	43.5	1	Lock prob
10	89.6	123.2	1	Buffer prob

(b) Annotated failure data L

time	lock_time	num_io	failures
6	75.6	83.5	1
7	72.4	83.8	1

(c) Unannotated failure data U

time	lock_time	num_io	failures
14	70.0	80.7	1
15	71.9	85.6	1

(d) Failure data F

Fig. 1. Sample monitoring data

are recurrences of known problems, i.e., those whose cause has already been ascertained or is under investigation. The implication is that if we do diagnosis from scratch for each observed failure, then we are likely to incur a terrible waste of past diagnosis effort.[1]

To exploit both the above observations, we can first check whether F matches the data from a previously-diagnosed failure in D. Simple solutions seem to apply, e.g., find the nearest neighbor(s) of F in D. However, there is a nontrivial hurdle to be overcome. Failure data in real systems contains various types of *errors*: (i) natural system variability injects Gaussian noise; (ii) failures often corrupt readings; and (iii) rapid system state transitions cause readings from different states to get mixed up.

These errors, where the true value differs from the observed value in arbitrary ways, mislead conventional solutions; which motivated us to develop a new solution that generates a *robust* database of *failure signatures* from the historic monitoring data D. This database can match F with the right failure signature even when D and F contain some degree of error. We demonstrate that our signature database remains competitive and has much slower drop of accuracy compared to competing techniques as error in the monitoring data increases.

A. Our Contributions

The work reported in this paper was done in the context of the *Fa* system for mining the large volumes of high-dimensional and noisy monitoring data generated by databases and other systems. Fa periodically collects monitoring data from a running system, and stores this data in a relational database. The data has schema of the form $\tau, x_1, x_2, \ldots, x_n$, where τ is the monitoring time interval, and each x_i is a system performance metric. Over 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.[2]

Example 1.1: Figure 1 displays the historic data collected by monitoring a database server at one-minute intervals. In each monitoring interval (`time`), attribute `lock_time` is the average wait time to acquire locks; `num_io` is the number of disk I/Os; `failures` denotes whether the average response time of database transactions in that interval exceeded a threshold (causing SLO violations) or not; and `annotation` records the cause of each diagnosed failure.

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.

This paper identifies two new problems posed by the processing of diagnosis queries, and develops efficient and fairly-accurate algorithms for these problems.

Problem I (Diagnose(F, H)): Can the cause of the failure be characterized succinctly as attributes whose values in F deviate from their values in the data representing the healthy states of the system?

Solution and Challenges: Figure 2(b) illustrates Fa's solution to this problem. We have developed anomaly-based clustering that gives special attention to F while clustering H, to identify a minimal set of healthy system states needed for accurate diagnosis of F (i.e., low false positives and negatives). Anomaly-based clustering outperforms classic (K-means [22]) and recent clustering algorithms (LAC [11]) on both efficiency and diagnosis accuracy. On high-dimensional and noisy monitoring data, Fa produces very concise attribute sets that characterize the deviation of F from healthy states.

Problem II (Diagnose(F, L)): Is the failure represented by F the same as a previously-diagnosed failure in L?

Solution and Challenges: Figure 2(a) illustrates Fa's solution to this problem. A database of failure signatures *SD* is generated offline from L. The signature *sig* in *SD* that matches F the best is found, along with a confidence estimate for this match. If the confidence estimate is larger than a threshold, then the annotation associated with sig is returned as the root cause of the failure. Two important challenges that we address are (i) dealing with errors in the failure data, and (ii) generating reliable confidence estimates and the threshold *automatically*.

Figure 2 illustrates how Fa combines the solutions for Diagnose(F, L) and Diagnose(F, H) to process a given diagnosis query. Diagnose(F, L) is done first since it is efficient (most of the work is offline) and has a high chance of finding the real root cause of the failure. (Recall that 50-90% of

[1] [5] is from IBM; similar sentiments have been echoed by practitioners of Oracle and Microsoft SQL Server.

[2] The use of U in Fa is the subject of a separate paper [14].

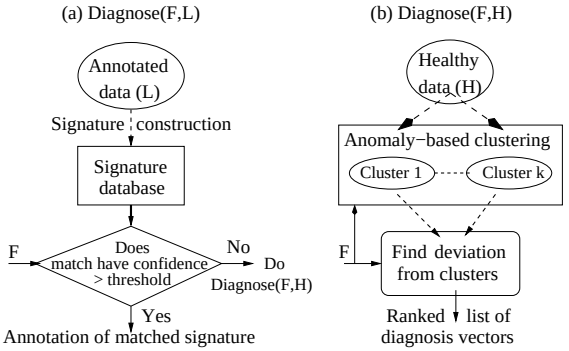

Fig. 2. Control and data fbw in Fa

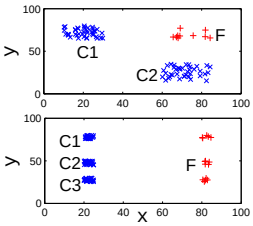

Fig. 3. Plots (a) and (b) of sample data

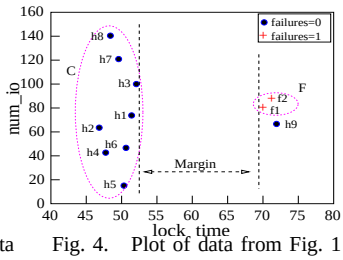

Fig. 4. Plot of data from Fig. 1

failures are recurrences of previously-diagnosed failures [5].) If Diagnose(F, L) cannot produce a match with confidence greater than the automatically-generated threshold, then Fa invokes the general-purpose, but costlier, Diagnose(F, H). Note the importance of reliable confidence estimates and the threshold: low thresholds may reduce diagnosis accuracy while high thresholds may invoke Diagnose(F, H) unnecessarily.

We will begin in Section II by describing Fa's solution for Diagnose(F, H). Section III describes Fa's solution for Diagnose(F, L). Section IV demonstrates the effectiveness of Fa through a comprehensive experimental evaluation based on failures from a production setting, a variety of failures injected in a testbed, and synthetic data.

II. Processing Diagnose(F,H) using Anomaly-based Clustering

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 first illustrate the main ideas using a series of examples. Suppose H consists of the instances in Figure 3(a) shown using the "x" symbol. Each instance has two attributes, x (denoting `lock_time`) and y (denoting `num_io`), plotted along the horizontal and vertical axes respectively. 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: (i) C_1 with $x \in [10, 30]$ and $y \in [65, 80]$, differing from F primarily along the x attribute; and (ii) C_2 with $x \in [60, 80]$ and $y \in [15, 30]$, differing from F primarily along the y attribute. A conventional clustering algorithm like K-means or LAC [11] 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 3(b). A conventional clustering algorithm will now group the instances in H into three distinct clusters (C_1, C_2, and C_3 in Figure 3(b)). 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(b) indicates that this answer is incorrect. Both the failure data and the 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? While clustering H, conventional clustering algorithms ignore how the instances in H differ from the failure instances F. 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 the system administrator.[3] Section IV-B validates these observations empirically.

We have developed a new clustering methodology, called *anomaly-based clustering*, where H is clustered with consideration of the instances F to be diagnosed.[4] (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(b), generating a single cluster for H, and linking the failure to attribute x only. We now describe anomaly-based clustering.

A. Diagnosis Vectors and Margin Classifiers

Fa processes a *Diagnose(F, H)* query by first clustering the healthy data H into a set of clusters C_1, C_2, ..., C_l, and then outputting the deviation of F from these clusters in the form $\{\langle \vec{w}_1, C_1\rangle, \langle \vec{w}_2, C_2\rangle, \ldots \langle \vec{w}_l, C_l\rangle\}$ as the diagnosis result. l depends on the query, and is not a predetermined constant. $C_1 \cup C_2 \cup \cdots \cup C_l$ need not include all the instances in H. Thus, outlier instances in H will be ignored.

Each $\vec{w} \in \{\vec{w}_1, \ldots, \vec{w}_l\}$ is called a *diagnosis vector*. $\vec{w}$ has the form: $\vec{w} = \langle w_1, w_2, \ldots, w_n\rangle$, where each attribute $x_j \in \langle x_1, x_2, \ldots, x_n\rangle$ is given a weight w_j such that $-1 \leq w_j \leq 1$ and $\sum_{j=1}^{n} |w_j| = 1$. Intuitively, $\vec{w}_i \in \{\vec{w}_1, \ldots, \vec{w}_l\}$ specifies the weighted list of attributes to which the failure can be localized by comparing the instances in C_i to the failure instances F. From a usability perspective, C_i serves as the evidence why $\vec{w}_i$ is reported in the diagnosis result.

Computing the Diagnosis Vector: Since we are dealing with high-dimensional data, a most desirable property of each $\langle \vec{w}, C\rangle$ is to make $\vec{w}$ as concise as possible. That is, the weights of all attributes that do not help differentiate between C and F should be zero. This property enables the system administrator to zoom in quickly on likely causes of the failure without being misled by false positives. Fa achieves this property [16] by finding the linear combination $\sum_{j=1}^{n} w_j x_j$ of attributes $\langle x_1, \ldots, x_n\rangle$ that produces the maximum separation between C and F. This maximum separation is called the *margin* between C and F.

Example 2.1: Consider query *Q=Diagnose*(F, H) from Example 1.1 and Figure 4. Let $C = H - \{h9\}$. ($h9$ is an

[3] Semi-supervised or constraint-based clustering do not solve these problems; see Section V.

[4] We assume that instances in F belong to the same type of failure.

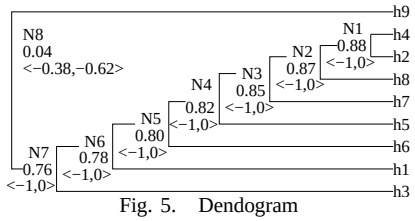

Fig. 5. Dendogram

erroneous observation generated while the system transitioned from a healthy state to a failure state.) The margin between C and F is produced between the two dotted lines in Figure 4: the line `lock_time` = 51.9 and the line `lock_time` = 70.0. Thus, $margin = 18.1$. Since the margin is produced along `lock_time`, the diagnosis vector $\vec{w} = \langle w_1, w_2 \rangle$ (corresponding to $\langle$`lock_time`, `num_io`$\rangle$) that produces the margin is w_1=−1 and w_2 = 0. $\langle 1, 0 \rangle$ also produces the same margin.

We can write a linear program to compute (i) the margin between C and F, and (ii) the diagnosis vector that produces the margin. We call this program a *margin classifier*, denoted *MC*(F, C) [16]. Section IV-B.3 empirically validates how *MC*(F, C) produces concise and correct diagnosis vectors.

B. Strawman: Margin-based Agglomerative Clustering

We begin with a strawman algorithm, called *margin-based agglomerative clustering (MAC)*, for anomaly-based clustering of H. MAC was proposed originally in [16] for analyzing cancer-related microarray data, and we have extended it to process diagnosis queries. The distinctive feature of MAC is an *agglomerative hierarchical clustering* [22] of the instances in H where the margin from the failure instances F is used as the metric for clustering. Note that conventional clustering schemes use distance-based metrics like Euclidean distance.

MAC starts by placing each instance in H in its own active cluster. In each successive iteration, *MC*$(C_i \cup C_j, F)$ is computed for each pair $\langle C_i, C_j \rangle$ of clusters among the remaining active clusters. MAC then picks the cluster pair $\langle C_{i'}, C_{j'} \rangle$ that gives the maximum margin with respect to F, and merges them together to form a single combined cluster. The merged clusters $C_{i'}, C_{j'}$ are no longer considered active. This process is repeated until all instances are merged into a single cluster. The sequence of cluster merges can be represented as a *dendogram*, which is a tree with the instances in H as leaves, and each new cluster as a nonleaf node.

Example 2.2: Figure 5 shows the dendogram generated by MAC for the healthy data in Example 1.1 and Figure 4. The margin (computed after normalizing the data) and diagnosis vector for the cluster at each nonleaf node are also shown.

We can generate clusters from the dendogram by selectively deleting nonleaf nodes, which will partition the dendogram into a forest of trees. The instances comprising the leaves of each tree form a cluster C that will be output as a $\langle \vec{w}, C \rangle$ pair in the diagnosis result (after computing $\vec{w}$ using *MC*(F, C)).

Consider a node P in the dendogram with child nodes L and R. Let the clusters corresponding to these three nodes be C_p, C_l, and C_r respectively. (Note that C_l and C_r were merged to form C_p in the dendogram, i.e., $C_p = C_l \cup C_r$.) Let the margin and corresponding diagnosis vector for these three clusters be $\langle m_p, \vec{w}_p \rangle$, $\langle m_l, \vec{w}_l \rangle$, and $\langle m_r, \vec{w}_r \rangle$ respectively. Note that $\vec{w}_p$ is the diagnosis vector that gives the margin for the combination of C_l and C_r. Intuitively, if the margin of C_l (C_r) along $\vec{w}_p$ is significantly less than the margin of C_l (C_r), then merging C_l with C_r will dilute the "clusteredness" of C_l (C_r) with respect to the failure instances F.

Exploiting this intuition, we will delete P if any of the following conditions is satisfied: (i) *Margin*$(C_l, F, \vec{w}_p) < (1 - \alpha)m_l$, or (ii) *Margin*$(C_r, F, \vec{w}_p) < (1 - \alpha)m_r$. Here, *Margin*$(C, F, \vec{w})$ denotes the margin (separation) of a cluster of instances C from the failure instances F along the diagnosis vector $\vec{w}$. α is a small positive constant, e.g., $\alpha = 0.2$.

Example 2.3: When the dendogram in Figure 5 is partitioned, the nonleaf node N_8 will be deleted, to generate two output clusters $\{h1, h2, h3, h4, h5, h6, h7, h8\}$ and $\{h9\}$; precisely what we expect based on Figure 4.

C. Partition-Check-Merge (PCM) Algorithms

MAC can give good diagnosis accuracy, but it scales poorly with $|H|$. Note that MAC requires $O(|H|^2)$ invocations of *MC* since MAC starts by invoking *MC*$(\{t_1, t_2\}, F)$ for every pair of instances t_1, t_2 in H.

We now propose an algorithmic framework that combines the good features of MAC (accurate, but inefficient) with those of conventional Distance-based Partitional[5] Clustering (*DPC*) algorithms like K-means and LAC (efficient, but less accurate). This new framework is called *Partition-Check-Merge (PCM)* because it has the following structure:

1. One or more *partitioning phases* that use an efficient DPC algorithm to partition the data progressively into more and more clusters until the check in Step 2 is satisfied. This progressive cluster refinement is achieved by increasing the input parameter k to the DPC algorithm that specifies the number of clusters to generate.
2. One or more *checking phases* that perform checks, namely, evaluating the current partitioning of instances to see whether this partition is good enough to be the set of clusters produced during an intermediate stage of MAC. If a check succeeds, then PCM moves to the merging phase; otherwise, partitioning is redone, possibly with a larger k.
3. A *merging phase* where the current set of clusters are merged progressively, like in MAC, to possibly consolidate several small clusters into a minimal set of clusters (representing diagnosis vectors and evidence) that can be output in the diagnosis result.

Observation: A degenerate case of PCM is one where the check never succeeds, so the partitioning phase eventually places each instance into a separate cluster. In this case, the merge phase will resemble running MAC from scratch.

However, for most monitoring datasets, the check will succeed much earlier—e.g., once k becomes equal to or larger than the best k for the data—avoiding the $O(|H|^2)$ complexity of MAC. If the partitioning phase generates more clusters than optimal, then the merging phase will glue back clusters that

[5]Intuitively, partitional algorithms work in a top-down fashion, while agglomerative algorithms work bottom-up.

should not have been split; at some loss of efficiency. In effect, PCM can be as accurate as MAC, while leveraging the efficiency of DPC. The challenge in PCM is in the implementation of the check phase. Next, we discuss two concrete instantiations of PCM.

D. PCM-Conservative (PCM-C)

PCM-Conservative (PCM-C) (Figure 6) uses a conservative implementation of check to process a *Diagnose*(F, H) query. For each instance $t \in H$, PCM-C first computes m_t, the individual margin between t and the failure instances F.

PCM-C uses an efficient DPC algorithm to partition the data instances into clusters. (For example, Fa uses the LAC algorithm [11] in Line 7 in Figure 6.) Suppose the clusters $\{C_1, \ldots, C_k\}$ are produced by a partitioning step. For each $C \in \{C_1, \ldots, C_k\}$, let $\langle m_C, \vec{w}_C \rangle$ be the margin and corresponding diagnosis vector for C and F. PCM-C's check phase lists an instance $t \in C$ as *covered by* C if *Margin*$(\{t\},F,\vec{w}_C) \geq (1-\alpha)m_t$. (Recall from Section II-B that *Margin*$(\{t\}, F, \vec{w}_C) \leq m_t$, since m_t is t's maximum margin across all vectors.) That is, t is covered by the cluster C that t was assigned to by DPC if t's margin along C's diagnosis vector is close enough to t's individual margin.

If t is covered by C, then (i) t will not be considered again during partitioning (Line 21), and (ii) t will be associated with C in the input to the merge phase. PCM-C iterates through the partitioning and merge phases until all instances get covered. If check finds that the current set of clusters $\{C_1, \ldots, C_k\}$ do not cover a significant fraction of the remaining instances, then partitioning is redone with a larger k; currently, we double k when this situation arises (Lines 16-17). Thus, while PCM-C starts with a small default value of k, k will get incremented automatically if required.

Once a set of DPC-generated clusters have covered all instances in H, these clusters are input to the merge phase. Merge does MAC-style agglomerative clustering—starting with these clusters as the leaves of the dendogram, instead of the $|H|$ individual instances—to generate the diagnosis result.

E. PCM-Eager (PCM-E)

We found empirically (Section IV-B) that PCM-C tends to generate many clusters as input to the merge phase. While merge can glue back clusters split more than needed, PCM-C's merge remains inefficient because of the quadratic dependence on the number of input clusters. PCM-E tackles this problem.

Recall that PCM-C's check phase will list a instance t as covered only if t's margin along the diagnosis vector of the cluster C that t was assigned to by DPC is close enough to t's individual margin. *PCM-Eager (PCM-E)* relaxes this condition as follows: PCM-E's check lists t as covered if t's margin along the diagnosis vector of *any* of the clusters generated by DPC so far is close enough to t's individual margin. As before, if t is covered by C, then t will be assigned to C in the input to the merge phase. (Note that DPC may not have assigned t to C.) Intuitively, PCM-E reduces DPC's role to identifying significant diagnosis vectors $\vec{w}$ from the data. The *Margin*$(\{t\}, F, \vec{w}) \geq (1-\alpha)m_t$ condition is used to associate instances with each $\vec{w}$, creating clusters that are input to the merge phase. The rest of PCM-E is similar to PCM-C.

Algorithm *Partition-Check-Merge-Conservative (PCM-C)*
Input: *Diagnose*(F, H) query; a Distance-based Partitional Clustering algorithm (DPC) like K-means/LAC; α (default is 0.2)
Output: Result in the $\{\langle \vec{w}_1, C_1 \rangle, \langle \vec{w}_2, C_2 \rangle, \ldots \langle \vec{w}_l, C_l \rangle\}$ format
/* First compute the individual margin of each instance $t \in H$ */
1. Compute $\langle m_t, \vec{w}_t \rangle = MC(\{t\}, F)$ for each instance $t \in H$;
2. k = default value; /* DPC's number of clusters parameter */
3. $Rem\_pts = H$; /* instances not assigned to clusters yet */
4. *Coverings* $= \phi$; /* assignment of instances to clusters */
5. **While** ($Rem\_pts \neq \phi$) {
6. /* Partitioning phase */
7. Partition Rem_pts with DPC into clusters $\{C_1, \ldots, C_k\}$;
8. *Outliers* $= \phi$;
9. /* Check phase: Lines 10-22 below */
10. **For** (each instance $t \in Rem\_pts$) {
11. Let C_i be the cluster that t was assigned to by DPC;
12. **If** (*Margin*$(\{t\}, F, \vec{w}_{C_i}) \geq (1-\alpha)m_t$)
13. Mark t as covered by C_i;
14. **Else** Add t to *Outliers*;
15. } /* end for */
16. **If** ($\frac{|Outliers|}{|Rem\_pts|} > 0.9$)
17. $k = k \times 2$; /* increase k, Rem_pts is unchanged */
18. **Else** {
19. **For** (each cluster $C_i \in \{C_1, \ldots, C_k\}$ that covers instances)
20. Add C_i and the instances C_i covers to *Coverings*;
21. $Rem\_pts =$ *Outliers*; /* remove covered instances */
22. } /* end else */
23. } /* end while */
24. /* Merge phase */
25. Initialize a partially-built dendogram with the clusters in
26. *Coverings* as the leaves of the dendogram;
27. Proceed with MAC using this partially-built dendogram;

Fig. 6. PCM-Conservative (PCM-C)

F. Filtering and Ranking Diagnosis Results

Fa takes the set of $\langle \vec{w}_i, C_i \rangle$ pairs generated by anomaly-based clustering, and outputs the final diagnosis result as a filtered and ranked list.

- **Filtering:** Fa removes $\langle \vec{w}_i, C_i \rangle$ pairs where the cluster size $|C_i|$ does not satisfy a minimum support threshold; similar to support thresholds in frequent-itemset mining. Clusters composed of outliers get eliminated here.
- **Ranking:** The remaining $\langle \vec{w}_i, C_i \rangle$ pairs are ranked in decreasing order of cluster size $|C_i|$.

III. Generating and Using a Robust Signature Database to Process Diagnose(F,L)

So far we described how Fa implements the baselining-based approach to diagnosis. We now move on to how Fa checks whether the failure to be diagnosed is the same as a failure that has already been diagnosed in the past (thus its root cause is known). We are given L, the subset of historic data with annotations about m distinct failures, denoted A_1, A_2, …, A_m. Fa generates a *signature database* from L that contains entries of the form $\langle sig, A_i \rangle$ where sig is a signature for failure A_i. Instances F from an undiagnosed failure can be matched against this database (recall Figure 2(a)).

We will first illustrate our ideas using a series of examples. Suppose L is as shown in Figure 7(a). Each instance has two

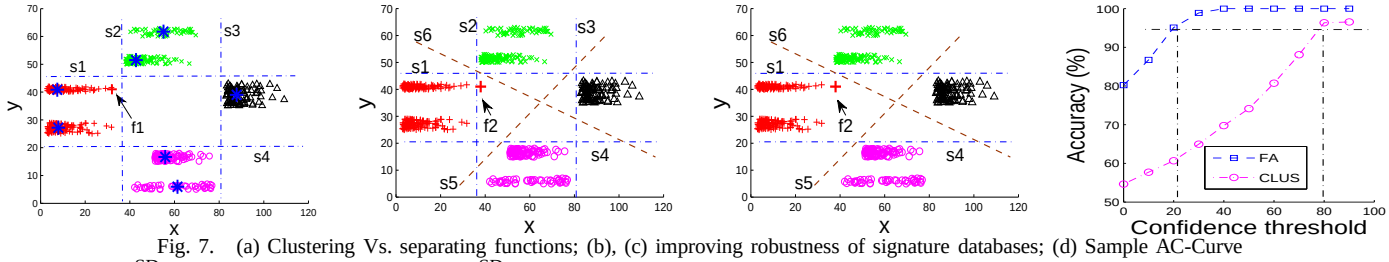

Fig. 7. (a) Clustering Vs. separating functions; (b), (c) improving robustness of signature databases; (d) Sample AC-Curve

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

SD_3

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

s_1 = 1 if y > 45, otherwise 0
s_2 = 1 if x < 38, otherwise 0
s_3 = 1 if x > 80, otherwise 0
s_4 = 1 if y < 20, otherwise 0
s_5 = 1 if 0.81*x − y > −16, otherwise 0
s_6 = 1 if 0.36*x + y > 61, otherwise 0

Fig. 8. Signature databases: (a) SD_2, (b) SD_3

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 7(a) shows the clusters per failure type. The centroids of these clusters—represented by blue stars ("⋆") in Figure 7(a)—become the signatures for the corresponding failure type, giving a signature database SD_1.

Suppose $f_1 = \langle 32, 41\rangle$ in Figure 7(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 7(a) that f_1 is an instance of Failure A_2 (plus). This example shows the drawbacks of clustering-based signatures. (They work poorly on real data in our experiments.)

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. These functions can take many forms. To convey our ideas while keeping the example simple, we will use a simple form, namely, separating lines in the 2D plane. Figure 7(a) shows s_1–s_4 as dotted lines. 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 8(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. For example, $\vec{s}(32, 41)$ is $\langle 0, 1, 0, 0\rangle$ for $f_1 = \langle 32, 41\rangle$. ($\langle 0, 1, 0, 0\rangle$ matches Failure A_2's signature perfectly.) For now, we will measure distances in terms of the *Hamming distance*, namely, the number of bits that are different. Thus, the distances of $\vec{s}(32, 41)$ to the four signatures in Figure 8(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 7(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 8(b) to diagnose f_2. SD_3 contains two new separating functions: $s_5(x, y)$, $s_6(x, y)$. s_5 separates instances of Failures A_1 and A_2 from those of A_3 and A_4, and s_6 separates instances of A_1 and A_3 from those of A_2 and A_4; as represented by the s_5 and s_6 columns in Figure 8(b). The separating planes for s_5 and s_6 are shown in Figure 7(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 8(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. The idea here is to transmit some selected extra bits along with regular data so that the receiver can reconstruct the original data in the presence of errors caused by noise or other impairments during transmission.

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 ≥ 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. However, the Hamming distance between any two signatures in SD_2 is ≥ 2. Thus, a 1-bit error in $\vec{s}(39, 41)$ leaves it in an ambiguous position for SD_2; causing incorrect diagnosis.

The previous example shows that selected redundancy in the set of separating functions can overcome incorrect predictions by some of the functions. Learning more functions increases the cost of generating the signature database. However, this work is mostly done offline, and can be made efficient with parallel learning of functions. The more pressing issue is that some functions are less reliable than others, and their presence can hurt diagnosis accuracy (and confidence) significantly.

s_2 and s_3 are two less reliable functions in our example. Note that the data for each type of failure in Figure 7(a) shows larger spread along the x axis than the y axis. Intuitively, there are larger chances of error in the x values. Since s_2 and s_3 separate exclusively along the x axis, they are likely to get their predictions wrong when errors in x arise (like what happened for f_2). We can drop s_2 and s_3, and generate a new

signature database SD_4 that has the functions s_1, s_4, s_5, and s_6 only (Figure 7(c)). SD_4 gives a perfect match for f_2.

Takeaway points: Our series of examples show that the following is a powerful representation of the signature database to achieve both good accuracy and robustness to errors:

- *Binary matrix M:* The ith row in M, denoted $M(i,:)$, is the signature of failure $A_i, 1 \leq i \leq m$. The jth column in M, denoted $M(:,j)$, corresponds to the separating function $s_j(\vec{x})$. The number of columns d depends both on m and the built-in error tolerance desired.
- *Separating functions $s_1(\vec{x})$–$s_d(\vec{x})$:* Each function separates one or more types of failure instances from the others. These functions can take many different forms, e.g., *Classification and Regression Trees (CART)* [22].
- *Weights β_1–β_d for the respective functions:* For robustness to errors, the prediction $s_j(\vec{x})$ from a less reliable separating function s_j is given a smaller weight while computing the distance of $\vec{s}(\vec{x})$ to the signatures. For example, reasonable weights for s_1–s_6 in SD_4 are $\{1, 0, 0, 1, 1, 1\}$ because s_2 and s_3 are less reliable.

Next, we describe the offline generation of the signature database, its use for diagnosis, and online maintenance.

A. Generating the Binary Matrix

There are four rules to generate a valid matrix M:

(I) Each row should be distinct since no two failures can have the same signature.

(II) Columns that contain all 0s or 1s should be excluded, since they do provide no differentiation among failures.

(III) Two columns cannot be the same or complementary since they derive the same separating function. (A 0-1 exchange in a column generates its complementary.)

(IV) The *radius* r of M, defined as half the minimum Hamming distance over all $\langle M(i,:), M(j,:)\rangle$ pairs, $i \neq j$, should be above a given threshold. Intuitively, the higher the radius, the higher the error-correction ability of M. For a failure instance $\vec{x}$, $\vec{s}(\vec{x})$ can be matched with the correct signature even when up to $r-1$ separating functions produce wrong predictions for $\vec{x}$.

We use a random search algorithm to generate M given a threshold R_t on M's radius. The algorithm is as follows:

1. Generate m random binary vectors of length $d = R_t(2+\delta)$. δ is a positive integer (we set δ =1). The expected Hamming distance between each pair of vectors is $d/2$.
2. Remove columns containing all 0s or 1s (Rule II).
3. For any identical or complementary column pair, retain one column only (Rule III).
4. If the matrix generated by Steps 1-3 has a radius smaller than the threshold R_t, then go to Step 1.

This simple algorithm is surprisingly effective. The radius threshold R_t is a design choice best left to the administrator. R_t balances diagnosis accuracy and robustness against the time to generate the signature database—higher R_t means more columns (functions), and hence longer time to generate the database. Based on our empirical observations, $R_t = 5log_2(m)$ is a balanced choice, and is Fa's default.

B. Generating the Separating Functions

For each column $M(:,j)$, Fa learns the separating function $s_j(\vec{x})$ as a binary classification tree (CART) separating the instances with annotation $M(i,j) = 1$ from the instances with annotation $M(i,j) = 0$. In separate work [13] with many types of functions from statistical machine-learning, we found CARTs to best balance prediction accuracy and learning time.

C. Weighting the Separating Functions

Suppose the jth separating function $s_j(\vec{x})$ has weight β_j, and the overall weight vector is $\vec{\beta} = \langle \beta_1, \beta_2, \ldots, \beta_d \rangle$. A failure instance $\vec{x}$ whose true annotation is A_i will be matched with A_i's signature $M(i,:)$ if the following condition holds:

$$\min_{k \neq i} \{\texttt{Dist}(\vec{s}(\vec{x}), M(k,:); \vec{\beta}) - \texttt{Dist}(\vec{s}(\vec{x}), M(i,:); \vec{\beta})\} > 0 \quad (1)$$

Here, Dist is the *weighted* Euclidean distance: $\texttt{Dist}(\vec{u}, \vec{v}; \vec{\beta}) = \sqrt{\sum_{j=1}^{d} \beta_j (u_j - v_j)^2}$. For binary vectors $\vec{u}$ and $\vec{v}$, the Euclidean distance is the square root of the Hamming distance.

The larger the difference in Equation 1, the higher the chances of matching $\vec{x}$ to the correct signature when errors cause variations in $\vec{s}(\vec{x})$. Thus, to make the signature database robust, we want to choose the weight vector $\vec{\beta} = \langle \beta_1, \ldots, \beta_d \rangle$ that maximizes this difference over all instances $\langle \vec{x}, A_i \rangle$ in the annotated failure data L. Under the condition that $||\vec{\beta}||^2 (= \sum_{j=1}^{d} \beta_j^2)$ is fixed, this optimization is:

$$\max_{\vec{\beta}} \min_{\langle \vec{x}, A_i \rangle \in L, k \neq i} \{\texttt{Dist}(\vec{s}(\vec{x}), M(k,:); \vec{\beta}) - \texttt{Dist}(\vec{s}(\vec{x}), M(i,:); \vec{\beta})\}$$

This optimization problem can be mapped to the problem of learning *Support Vector Machines (SVMs)* [22]. Fa uses an SVM learning algorithm from [20] to solve this problem and learn the weights $\vec{\beta}$; details are in the technical report [15].

D. Online Use and Maintenance

So far we discussed how the signature database is generated offline from L. We now discuss the online use of the database for diagnosis, and its maintenance as new instances and annotations are added to L. When the database is queried with an undiagnosed failure instance $\vec{x}$, Fa first computes $\vec{s}(\vec{x}) = \langle s_1(\vec{x}), \ldots, s_d(\vec{x}) \rangle$, and then finds the signature nearest to $\vec{s}(\vec{x})$, namely, the signature that minimizes $\texttt{Dist}(\vec{s}(\vec{x}), M(i,:); \vec{\beta}), 1 \leq i \leq m$. As shown in Figure 2(a), a confidence estimate *conf* is generated for this match and compared with the *confidence threshold* C_t. If *conf* $\geq C_t$, then the annotation of the matched signature is returned as the diagnosis result; otherwise diagnosis proceeds to *Diagnose*(F, H).

Confidence estimate: When $\vec{x}$ is matched with the signature $M(i,:)$, we define the confidence in this match as:

$$\textit{conf} = \min_{k \neq i} \{\texttt{Dist}(\vec{s}(\vec{x}), M(k,:); \vec{\beta}) - \texttt{Dist}(\vec{s}(\vec{x}), M(i,:); \vec{\beta})\} \quad (2)$$

Intuitively, *conf* is high when the second-nearest neighbor N_2 of $\vec{s}(\vec{x})$ is far from the first-nearest neighbor N_1, indicating an unambiguous match to N_1. As the gap between N_1 and N_2 shrinks, the ambiguity in the match to N_1 increases, and the confidence decreases.

To make the confidence estimate easier for administrators to understand, Fa converts it into a value in $[0, 100]$. This conversion is done using an equi-depth histogram (quantiles) with 100 buckets generated from the distribution of confidence estimates over all instances of L. Let p_k, $0 \leq k < 100$, denote the quantiles of this distribution. For a confidence estimate *conf* from Equation 2, Fa finds i such that $p_i \leq \mathit{conf} < p_{i+1}$; and reports i as the confidence estimate.

Setting the confidence threshold C_t: The value of C_t is critical because a low C_t can lead to incorrect diagnosis, while a high C_t can invoke the more expensive *Diagnose*(F, H) more often than needed. (Recall that compared to *Diagnose*(F, L), *Diagnose*(F, H) involves higher run-time overhead and more effort from administrators to interpret diagnosis results.) Fa's approach is to let the administrator specify the minimum diagnosis accuracy she wants from the signature database. Then, Fa automatically derives the appropriate C_t that gives this diagnosis accuracy while minimizing the chances of invoking *Diagnose*(F, H).

The main idea is to generate an *accuracy-confidence curve (AC-Curve)* for the signature database. Point x, y in this curve means when the confidence threshold is $C_t = x$, the signature database has an expected accuracy of $y\%$ for matches having confidence $\geq x$. Also note that when $C_t = x$, there is an $x\%$ chance that a *Diagnose*(F, L) query will return with a confidence $< x\%$; thus, the chances of invoking *Diagnose*(F, H) is $x\%$ per diagnosis query when $C_t = x$.

Figure 7(d) is a sample graph from our experiments which plots the AC-Curves for both Fa's signature database (FA) and the clustering-based signature database (CLUS). If the administrator desires a diagnosis accuracy of 95%, then Fa's $C_t = 20$ while CLUS's $C_t = 80$. That is, Fa is four times less likely to trigger *Diagnose*(F, H) than CLUS. We will revisit AC-Curves in the experimental evaluation in Section IV.

Fa has fully-automated procedures for generating the AC-Curve and confidence threshold C_t for a signature database, as well as maintaining all data structures incrementally as new instances and annotations are added to L. The details are given in the technical report [15].

IV. Experimental Evaluation

A. Experimental setting

We present a comprehensive experimental evaluation of Fa based on failures from a production setting, a variety of failures injected in a testbed, and synthetic data. Table I summarizes our datasets and failure scenarios.

Real failures in a production system: *Software aging*—the progressive degradation in performance caused by, e.g., memory leaks, unreleased file locks, and fragmented storage space—is a common cause of system failure [21]. The *Dolphin* and *ECE* datasets were collected from two production servers at Duke over the course of two months. This data records OS metrics at 10-minute intervals. Both servers crashed once or more during this period due to aging of different resources, as found by a previous study [21]. We validate Fa's automated diagnosis results with the results from this manual study.

Failures injected in a testbed: We have implemented a testbed that runs *Rubis* [19]—a multitier auction service modeled after eBay—on a JBoss application server and a MySQL DBMS. It is reported that software problems and operator errors are the common causes of failure in Web services [17]. We inject such failures into a running Rubis instance using a comprehensive failure-injection tool [6]. This setting makes it easy to study the accuracy of Fa's diagnosis algorithms because we always know the true cause of each failure.

Specifically, we can inject 3 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. Using this mechanism, we can inject 75 distinct single-EJB failures and any number of multiple-EJB failures (concurrent single-EJB failures). Intuitively, multiple-EJB failures are harder to diagnose. The *Rubis-bug*, *Rubis-jndi*, *Rubis-60*, and *Rubis-complex* monitoring datasets in Table I are from this setting. These datasets contain the number of times each distinct EJB procedure call is invoked per minute.

We can also inject failures caused by contention for CPU, memory, and disk resources. The *OLTP-single* and *OLTP-multi* monitoring datasets in Table I are collected from a MySQL DBMS running an OLTP workload, where we injected resource contention to cause failures. The datasets record OS metrics (e.g., CPU utilization, paging), DBMS performance counters (e.g., number of index accesses and table scans), and transaction-level performance metrics (e.g., average transaction response time) per minute.

Synthetic data: *Synthetic* is a complex dataset (*PENDIGITS*) from the UCI machine-learning repository.

B. Evaluation of Diagnose(F, H) Processing

Queries: We begin by evaluating the processing of *Diagnose*(F, H) queries. For datasets 1-6 listed in Table I, H contains the historic healthy monitoring data and F contains 5-10 instances from the listed failures. For Dolphin and ECE, F contains 5 instances collected just before each server's first crash. We consider two cases for OLTP-multi, one where F contains failure instances from CPU contention, and the other where F contains failure instances from disk contention. We can evaluate the accuracy of diagnosis results since the cause of failure in each case is known.

Algorithms and Defaults: We compare four algorithms: (i) MAC (Section II-B), (ii) PCM-C (Section II-D), (iii) PCM-E (Section II-E), and (iv) *LAC-Silhouette (LAC-S)*. LAC-S applies the LAC algorithm from [11] on H after computing the number-of-clusters parameter k that maximizes a validity index called *Silhouette* [4]. Silhouette aims to maximize the inter-cluster distances (the average distance of pairs of points from different clusters) and minimize the intra-cluster distances (the average distance of pairs of points from the same cluster). For each cluster $C \subseteq H$ generated by LAC, LAC-S outputs $\langle \vec{w}, C \rangle$ computed using $MC(F, C)$.

1) **Comparing Running Times:** Table II shows the running time of our algorithms on the different datasets. Each reported time was averaged over 10 runs. For LAC-S, the time

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 DATASETS USED IN THE EVALUATION. COLUMNS a AND i ARE THE NUMBER OF ATTRIBUTES AND INSTANCES RESPECTIVELY

shown is the time to compute silhouette indices for 10 different values of k. This time is an optimistic estimate of the running time of LAC-S because we expect that more than 10 choices of k will have to be explored before finding the k that maximizes the silhouette index. We currently try all values of $k \in [2, 30]$. The best k is reported in LAC-S's column in Table II.

Because of its poor scalability, we had trouble running MAC on the full version of all but the smallest dataset (OLTP-multi) in Table I. Therefore, the times for MAC are for scaled-down versions of the datasets. The scaled-down size is shown in MAC's column in Table II. The times for PCM-C and PCM-E are split into the time for the partitioning and checking phases, denoted T_p in Table II, and the time for the merge phase, denoted T_m. The following trends are clear in Table II.

- MAC is very inefficient because of $O(|H|^2)$ *MC* calls.
- PCM-E is by far the most efficient algorithm. Note that PCM-E's T_m is usually significantly better than that of PCM-C. This trend is because PCM-E's aggressive strategy to map points to clusters leads to a much lower number of clusters being input to the (quadratic) merge phase. PCM-E's T_p is also better because PCM-E's aggressive strategy gets all points covered in fewer iterations of the partitioning and checking phases.
- PCM-E typically matches or outperforms LAC-S, which is because the silhouette computations in LAC-S perform $O(|H|^2)$ distance computations.

2) **Comparing Diagnosis Accuracy:** Table III summarizes the diagnosis accuracy of our algorithms on the datasets. Numbers like 1st and 2nd in Table III indicate the smallest rank of a cluster C whose diagnosis vector gives non-zero weights to attributes relevant to the failure (therefore, smaller rank means higher diagnosis accuracy). Non-zero weights in this diagnosis vector are shown for PCM-E, with weights for attributes relevant to the failure shown in bold font. Each cell also shows the % size of C with respect to the number of historic points $|H|$, and the number of $\langle \vec{w}, C \rangle$ pairs in the result after filtering with a support threshold of 2%. The following trends are clear in Table III.

- PCM-C and PCM-E give the best accuracy in all cases.
- MAC gives good accuracy in most cases. Recall from Section IV-B.1 that MAC uses only a subset of the full historic data because of scalability problems.
- LAC-S gives poor accuracy in many cases.

As shown in Table III, the diagnosis accuracy of LAC-S is poor for the Dolphin dataset when we use $k = 19$ for which the silhouette cluster validity index is maximized. We tried LAC on this dataset with different values of k ranging from 2 to 30. In most cases, the relevant attribute—the attribute measuring available swap space, since the failure in Dolphin is because of swap space exhaustion—was not part of the diagnosis result. In the few cases where the relevant attribute was part of the diagnosis result, it appears in some lowly-ranked cluster (as for $k = 19$ in Table III) and/or as one among many attributes with nonzero weight in a diagnosis vector. Furthermore, as we increase k, LAC reports more and more clusters in the result, each cluster with its own diagnosis vector; so we cannot be confident about any of the output clusters or diagnosis vectors.

Dataset	LAC-S	MAC	PCM-C	PCM-E
Dolphin	260.3 k=19	5137.0 ($\|H\|$= 1000)	87.4 T_p=23.5 T_m=63.9	32.7 T_p=18.8 T_m=13.9
ECE	229.6 k=2	5187.9 ($\|H\|$= 1000)	89.1 T_p=18.9 T_m=70.2	26.1 T_p=17.2 T_m=8.9
Rubis-bug	28.3 k=3	1318.5 ($\|H\|$= 600)	45.8 T_p=40.1 T_m=5.7	34.7 T_p=28.8 T_m=5.9
Rubis-jndi	75.1 k=2	1399.6 ($\|H\|$= 600)	108.2 T_p=92.6 T_m=15.6	95.8 T_p=87.0 T_m=8.8
OLTP-single	214.5 k=2	5116.0 ($\|H\|$= 1000)	173.3 T_p=137.4 T_m=35.9	88.1 T_p=70.6 T_m=17.5
OLTP-multi,F= CPU contention	7.2 k=15	1526.1	7.1 T_p=4.3 T_m=2.8	3.8 T_p=3.2 T_m=0.6
OLTP-multi,F= Disk contention	6.2 k=14	1516.1	7.4 T_p=3.3 T_m=4.1	4.1 T_p=3.6 T_m=0.5

TABLE II
COMPARING RUNNING TIMES (SECONDS)

On the other hand, note from Table III that both PCM-C and PCM-E report the relevant attribute (as one of two attributes) in the diagnosis vector of the top-ranked cluster, which contains close to 40% of the total historic data. This result illustrates the power of PCM's anomaly-based clustering. The Dolphin data contains many patterns because of the general effects of software aging, causing LAC to generate clusters that are not relevant to the failure points to be diagnosed.

We have also compared PCM-E with two correlation-based techniques [7], [8] and two baselining-based techniques [3] proposed in previous work. The results of PCM-E are superior to all these techniques [15].

3) **Evaluation of PCM-E's Diagnosis Vectors:** The last column of Table III gives the diagnosis vector of the relevant

Dataset/Result	LAC-S	MAC	PCM-C	PCM-E	Diagnosis vector for PCM-E
Dolphin	8th, 5%, 14	Not found	1st, 39%, 8	1st, 37%, 7	2 nonzero weights, $(\mathbf{0.77}, \mathbf{0.23})$
ECE	Not found	2nd, 9%, 2	1st, 68%, 3	1st, 73%, 4	2 nonzero weights, $(\mathbf{0.75}, \mathbf{0.25})$
Rubis-bug	2nd, 25%, 3	1st, 14%, 7	1st, 19%, 3	1st, 21%, 6	2 nonzero weights, $(\mathbf{0.61}, 0.39)$
Rubis-jndi	Not found	1st, 43%, 2	1st, 21%, 2	1st, 48%, 5	1 nonzero weight, **1**
OLTP-single	Not found	2nd, 10%, 2	1st, 33%, 4	2nd, 26%, 5	4 nonzero weights, (**0.3**, 0.34, 0.11, 0.25)
OLTP-multi,CPU	3rd, 10%, 15	2nd, 25%, 4	2nd, 19%, 4	2nd, 26%, 5	3 nonzero weights, (**0.49**, 0.43, 0.08)
OLTP-multi,Disk	1st, 20%, 14	1st, 69%, 4	1st, 70%, 4	2nd, 74%, 4	3 nonzero weights, (**0.55**, 0.24, 0.19)

TABLE III
COMPARING DIAGNOSIS ACCURACY (SEE SECTION IV-B.2)

cluster produced by PCM-E. Note that PCM-E's diagnosis vectors have very few attributes of nonzero weight, even for high-dimensional datasets. This property makes it easy to interpret results. For example, PCM-E's diagnosis vector for Rubis-bug contains only 2 attributes (out of 110) with nonzero weights: one with weight 0.39, and the other with weight 0.61. (Sum of absolute weights is 1.) The latter attribute pinpoints the buggy Java bean. PCM-E's diagnosis vector for Dolphin contains only 2 attributes (out of 41) with nonzero weights: one with weight 0.77 and the other with weight 0.23; both pinpoint the swap space exhaustion problem causing the crash.

While there are no false negatives in PCM-E's diagnosis results (which is most important), there are some false positives in four cases. This problem shows the drawback of *Diagnose*(F, H) processing over *Diagnose*(F, L) processing: getting from the *Diagnose*(F, H) result to the root cause may require manual effort.

C. Evaluation of Diagnosis(F, L) Processing

Queries: We now consider *Diagnose*(F, L) queries over the Rubis-60, Rubis-complex, and Synthetic datasets. By default, L contains 60% of the failure instances in each dataset, and is used to generate the signature database. The remaining 40% of the failure instances are used to query the signature database to compute its diagnosis accuracy (% of times the correct annotation is returned).

Techniques: We compare three techniques: (i) CLUS, signature database implemented using K-means clustering with 10 clusters per annotation[6]; (ii) FA, Fa's signature database; and (iii) CART, a multi-class classifier implemented using classification and regression trees. By treating each annotation in L as a distinct class label, a multi-class classifier learned from L can predict the annotation of a new failure instance. We chose CART over other multi-class classifiers for three reasons: (i) CARTs are being used for diagnosis in production settings like eBay.com [7]; (ii) CARTs provide a principled way to compute confidence estimates; and finally, (iii) Fa uses CARTs as separating functions in its signature database.

1) **Comparing Accuracy-Confidence Curves:** The goal of *Diagnose*(F, L) is to provide high diagnosis accuracy while invoking *Diagnose*(F, H) only when required. The Accuracy-Confidence Curves (AC-Curves) in Figure 9 show how well each technique meets this goal. Recall the definition of confidence estimates, confidence thresholds, and AC-Curves from Section III. To diagnose a failure instance, CARTs compute a probability distribution over annotations, and output the most-likely annotation. The confidence estimate is the difference in probabilities between the first and second most-likely annotation, mapped to $[0, 100]$ as discussed in Section III-D.

[6]We also tried LAC clustering [11], and got similar results.

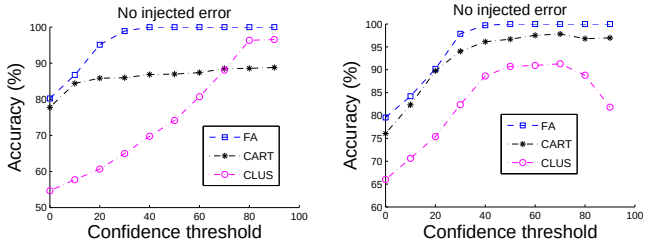

Fig. 9. AC-Curves for (a) Rubis-60, (b) Rubis-complex

Suppose the administrator wants a diagnosis accuracy of 90%. Then, Figure 9(a) for Rubis-60 shows that the confidence thresholds (C_t) for FA and CLUS can be set to 20 and 80 respectively. Since FA's C_t is 4 times less that of CLUS, FA is four times less likely to invoke *Diagnose*(F, H) at the same accuracy level. More interestingly, CART is unusable when required accuracy is 90%. FA maintains its superior performance for Rubis-complex, while CLUS now becomes unusable when the required accuracy is over 90%.

Figure 9 used our default setting where for each query instance $\langle \vec{x}, A \rangle$, the signature database contains at least one signature for annotation A. That is, we evaluated how good the signature database is in diagnosing previously-seen failures (which does not mean previously-seen instances). This setting is practical because up to 90% of all software failures reported by users today are previously-seen failures [5]. We now consider query instances whose correct annotations are not in the signature database. An accurate response from *Diagnose*(F, L) now is an answer with confidence below the threshold C_t; thereby forcing the invocation of *Diagnose*(F, H).

To create this setting, we divide the instances in Rubis-60 into two groups with nonoverlapping annotations: Group G_1 with 40 annotations and Group G_2 with the remaining 20 annotations. A subset of the instances from G_1 are used to construct the signature database. The remaining instances in G_1 (previously-seen failures) and the instances in G_2 (new failures) form the set of query instances. Figures 10(a) and (b) show the diagnosis accuracy of different techniques in these two cases. The behavior of signature databases (FA and CLUS) remain similar to Figure 9(a). However, CART performs poorly on the new failures, which shows a key advantage of using signature databases for *Diagnose*(F, L) rather than multi-class classifiers. Intuitively, signature databases have a

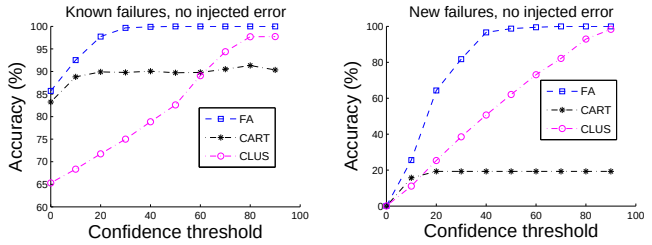

Fig. 10. AC-Curves for Rubis-60 with two groups: (a) existing annotations, (b) new annotations

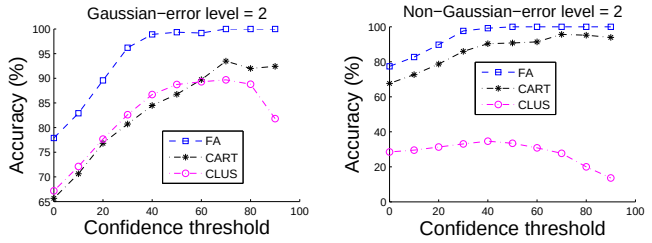

Fig. 11. AC-Curves for Rubis-complex: (a) Gaussian, (b) Non-Gaussian error

better chance of detecting when a failure does not have the symptoms of any previously-seen failure.

2) **Comparing Robustness to Error:** Our monitoring data contains two types of errors: *Gaussian* and *non-Gaussian*.

- Gaussian error is caused by natural variability in real systems. If we take multiple observations of an attribute from a particular system state, a goodness-of-fit test to a normal distribution will often be positive.
- Two main causes of error in our monitoring data cannot be modeled by Gaussian distributions: (i) the presence or onset of failure corrupts readings of some attributes (seen with JBoss application server and MySQL); (ii) readings from different states get mixed up in the same instance due to rapid system state transitions, or due to delays in measuring different attributes under system overload.

The query instances used so far to compute the accuracy of our diagnosis techniques were taken directly from the monitoring data. We now inject errors into these query instances to study how accuracy degrades as error increases. The technical report [15] gives the full details of error generation as well as our verification of the presence of Gaussian and non-Gaussian error in the monitoring data. Intuitively, we define an *error level* parameter $e \in [1, 4]$ that controls the scale of errors. For an attribute x: (i) Gaussian error of level e means that observations of x have a variance of $10e\%$ from their true value; and (ii) non-Gaussian error of level e means that each observation of x has a $10e\%$ chance of being an arbitrary value from the range of values of x.

Figure 11 shows the AC-Curves for level 2 of Gaussian and non-Gaussian error in Rubis-complex. (Note that Figure 9(b) is the AC-Curve at error level 0.) It is clear from comparing these graphs that the gap between FA and CLUS/CART increases as the error increases. CLUS is highly sensitive to non-Gaussian errors. Further evidence is provided by Figure 12 which shows the accuracy of different techniques as the error level, both for Gaussian and non-Gaussian, increases from 0 to 4 for Rubis-complex. (Rubis-60 has similar behavior [15].) These graphs validate FA's robustness to error.

3) **Scalability with Number of Annotations:** Figure 13 shows the trend as the number of annotations—i.e., different types of failure—is increased from 20 to 80. Since we can generate at most 75 distinct single-EJB failures (recall Section IV-A), the 80-failure dataset contains 5 multiple-EJB failures as well. The gap between FA and CLUS/CART increases as the number of failures increases.

In the offline phase of signature database generation, FA is less efficient than CART or CLUS. If the separating functions are not learned in parallel, then FA can take an order of magnitude more time to generate the database than CART or CLUS. However, these offline efforts make FA much better in the online phase because: (i) FA is comparable to CLUS and CART in the time for *Diagnose*(F, L); and (ii) FA invokes the more expensive *Diagnose*(F, H) much less often.

V. Related Work

Fa differs from previous work on automated diagnosis in three significant ways: (i) integration of *Diagnose*(F, L) (diagnosing recurrent failures) and *Diagnose*(F, H) (diagnosing failures not seen previously); (ii) considering robustness of diagnosis to errors in the monitoring data; and (iii) providing reliable estimates of confidence in diagnosis results.

Diagnose(F,H): Previous work on *Diagnose*(F, H) predominantly takes one of the *correlation-based* (e.g., [7], [8]) or *baselining-based* approaches (e.g., [3]). [7] applies decision-tree learning techniques to rank different system components based on their correlation with system failures. [8] applies Bayesian-network learning techniques to correlate performance metrics with high-level system behavior. [3] proposes a heuristic to represent the baseline behavior of a Web service; and applies anomaly detection techniques to categorize deviation from the baseline. We have compared PCM-E (anomaly-based clustering) empirically on real monitoring datasets with the algorithms in [7], [8], [3]. PCM-E outperformed the others due to the noisy and dynamic nature of the monitoring data. The experimental results are reported in [15].

Fa's *Diagnose*(F, H) processing is very different from semi-supervised clustering (using annotated data to improve clustering) and clustering with constraints [2]. We have to cluster H, and not $H \cup F$. Note that H has no annotations. (F has no annotations either, and typically, very few instances.) Moreover, our true benefits come from making clustering "diagnosis-aware" using a novel clustering metric that accounts for how F differs from instances in H. Recall from Section II-A that this metric is computed using a linear program (margin classifier). It is nontrivial to specify this metric as constraints or to incorporate it into a distance-based clustering algorithm like K-means (e.g., as in metric learning [2]).

Diagnose(F,L): Automated diagnosis of recurrent problems has been considered in previous work, e.g., [24], [5], [23], [9]. [24] builds a multi-class classifier on system event traces to classify previously-solved problems. We have empirically shown the advantages of signature databases over multi-class classifiers, especially in terms of robustness. [23] gives signature-generation techniques assuming that *symptoms* of each failure have been identified in the monitoring data; which

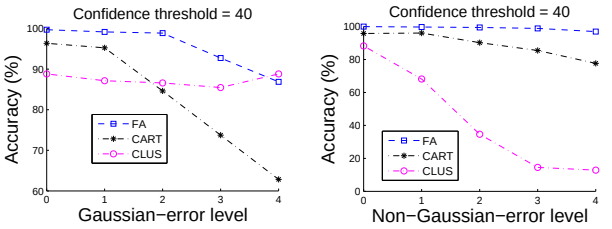

Fig. 12. Robustness curves for Rubis-complex: (a) Gaussian error, (b) non-Gaussian error

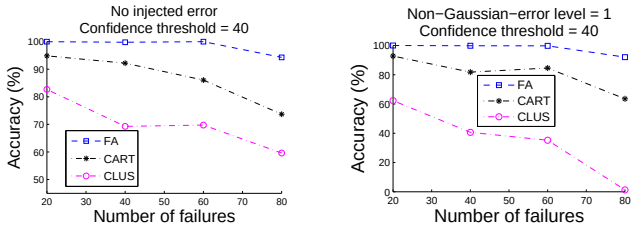

Fig. 13. Trend as the number of failures increases

is impractical in our settings where only raw monitoring data is available. [5] considers a very different type of monitoring data—function call stacks from system failures—and gives stack matching and indexing algorithms. [9] extracts indexable signatures from failure data by finding metrics that differentiate a failure state from the healthy states. Instead, Fa captures the difference of one or more failure states from other failure states using annotation information which is ignored in [9].

In effect, Fa's signature database solves a multi-class classification problem using an ensemble of binary classifiers (e.g., see [1]). However, with dynamic systems, high-dimensional monitoring data, and sparse failure data, there is a 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 the "test data" but not in the "training data". [25] shows the impact of such errors on accuracy. In *Diagnose*(F, L) processing, Fa is unique in its focus on robustness to such errors, reliable confidence estimates, and on the reliability and weighting of separating functions (Section III-C).

An early version of Fa [12] 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).

Failure diagnosis in database systems: Oracle's recent ADDM tool [10] shows the growing interest among database vendors in automated diagnosis of failures. Fa differs from ADDM in three ways: (i) ADDM relies on a static knowledge base gathered by Oracle experts over the years, while Fa automates the process of generating the signature database from monitoring data; (ii) Fa uses historic monitoring data in a principled way and enables reuse of past diagnosis results; and (iii) Fa targets a broader class of systems (e.g., multitier services). ADDM and Fa can complement each other.

VI. Conclusion

We showed how Fa makes five new contributions to address the challenges in building an automated diagnosis tool:

- *Noisy data:* Our empirical evaluation (Sections IV-C.2 and IV-B.2) showed how Fa maintains high diagnosis accuracy in the presence of errors in the monitoring data.
- *High dimensionality:* Even with 50-100 attributes, Fa identifies the 1-2 attributes related to a failure (Section IV-B.3).
- *Dynamic systems:* Both Fa's signature database generation and anomaly-based clustering can deal with multiple healthy and failure states and rapid state transitions.
- *Reuse:* Fa's signature database increases reuse of previous diagnosis results by giving high accuracy while minimizing costly invocations of *Diagnose*(F, H) (Section IV-C.1).
- *Trust:* Our empirical evaluation showed how Fa's confidence estimates and diagnosis vectors are very reliable.

References

[1] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *ICML*, 2000.

[2] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *ICML*, 2004.

[3] P. Bodik et al. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In *ICAC*, June 2005.

[4] N. Bolshakova and F. Azuaje. Cluster validation techniques for genome expression data. *Signal Processing*, 83(4), 2003.

[5] M. Brodie, S. Ma, G. M. Lohman, L. Mignet, N. Modani, M. Wilding, J. Champlin, and P. Sohn. Quickly finding known software problems via automated symptom matching. In *ICAC*, 2005.

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

[7] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *ICAC*, June 2004.

[8] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *OSDI*, Dec. 2004.

[9] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, Indexing, Clustering, and Retrieving System History. In *SOSP*, Oct. 2005.

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

[11] C. Domeniconi et al. Locally adaptive metrics for clustering high dimensional data. *Data Mining and Knowledge Discovery*, 14(1), 2007.

[12] S. Duan and S. Babu. Proactive identification of performance problems. In *SIGMOD*, 2006. Demonstration.

[13] S. Duan and S. Babu. Processing forecasting queries. In *VLDB*, 2007.

[14] S. Duan and S. Babu. Guided problem diagnosis through active learning. In *ICAC*, 2008.

[15] S. Duan, S. Babu, and K. Munagala. Automated Diagnosis of System Failures with Fa. Technical report, Mar. 2008. Available at http://www.cs.duke.edu/~shivnath/fa/diagnosis.pdf.

[16] K. Munagala et al. Cancer characterization and feature set extraction by discriminative margin clustering. *BMC Bioinformatics*, 2004.

[17] D. Oppenheimer, A. Ganapathi, and D. Patterson. Why do internet services fail, and what can be done about it. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

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

[19] *Rice University Bidding System*. rubis.objectweb.org.

[20] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML 2004*.

[21] K. Vaidyanathan and K. S. Trivedi. A comprehensive model for software rejuvenation. *IEEE Trans. on Dependable and Secure Comp.*, 2(2), 2005.

[22] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, June 2005.

[23] A. Yemini and S. Kliger. High speed and robust event correlation. *IEEE Communication Magazine*, 1996.

[24] C. Yuan et al. Automated known problem diagnosis with event traces. In *EuroSys*, 2006.

[25] X. Zhu and X. Wu. Class noise vs. attribute noise: A quantitative study. *Artif. Intell. Rev.*, 22(3):177–210, 2004.