Empirical Comparison of Techniques for Automated Failure Diagnosis

Songyun Duan and Shivnath Babu
Department of Computer Science, Duke University

Abstract

Automated techniques to diagnose the cause of system failures based on monitoring data is an active area of research at the intersection of systems and machine learning. In this paper, we identify three tasks that form key building blocks in automated diagnosis:

1. Identifying distinct states of the system using monitoring data.
2. Retrieving monitoring data from past system states that are similar to the current state.
3. Pinpointing attributes in the monitoring data that indicate the likely cause of a system failure.

We provide (to our knowledge) the first apples-to-apples comparison of both classical and state-of-the-art techniques for these three tasks. Such studies are vital to the consolidation and growth of the field. Our study is based on a variety of failures injected in a multitier Web service. We present empirical insights and research opportunities.

1 Introduction

Failures of Internet services and enterprise systems lead to user dissatisfaction and considerable loss of revenue. Manual diagnosis of these failures can be laborious, slow, and expensive. This problem has received considerable attention recently from researchers in systems, databases, machine learning and knowledge discovery, computer architecture, and other areas (e.g., [2, 5, 6, 7]). The research has been directed at developing automated, efficient, and accurate techniques for diagnosing failures using system monitoring data. These techniques are based on one or both of two fundamental primitives in machine learning: (i) *clustering*, which partitions data instances into groups based on similarity; and (ii) *classification*, which predicts the category of data instances based on attribute values.

While clustering and classification are well-studied problems, applying them in the context of automated diagnosis poses nontrivial challenges like:

- **High dimensionality:** Monitoring data routinely contain 100-1000 attributes, posing challenges from both an accuracy as well as running-time perspective.
- **Dynamic systems:** Modern systems are highly dynamic environments where workloads, resource availability, and system configuration vary over time, so failure patterns can change as well.
- **Noisy data:** Monitoring data from production systems contain 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.

This paper compares some classical as well as some more recent techniques that have been developed to address such challenges. For this comparison, we identify three tasks that form key building blocks in automated diagnosis. The tasks are presented in Section 2 along with the details of system monitoring data. The techniques we evaluate for each task are presented in Section 3. Our evaluation is based on an implementation of all techniques in the *Fa* system-management platform being developed at Duke [9, 10, 11].[1]

The evaluation setting is a testbed that runs the *Rubis* [13] auction Web service where we inject failures caused by factors like software bugs, data corruption, and uncaught Java exceptions. Section 4 presents the evaluation setting, methodology, and results. We conclude in Section 5.

2 Key Tasks in Automated Diagnosis

System Monitoring Data: When a system is running, Fa collects monitoring data periodically and stores it in a database. In this paper, we consider monitoring data having a relational (row and column) schema with attributes $X_1, \ldots, X_n$. For example, Fa collects more than a hundred performance metrics (e.g., average CPU utilization, number of disk I/Os) periodically from Linux servers. Application servers and database servers maintain performance counters (e.g., number of invocations of each Java module, 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.

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

- **Healthy data** H, which is monitoring data collected when the system was in a healthy state, i.e., when it experiences no violations of *service level objectives (SLOs)*.
- **Failure data** U, which is monitoring data collected from failure states of the system, i.e., when the system was not in a healthy state.

Apart from $X_1, \ldots, X_n$, the monitoring data contains an *annotation attribute* A. The value of A in an instance indicates whether the instance belongs to H or U. We consider three tasks that process the system monitoring data in different ways to aid failure diagnosis.

Task I: Identifying distinct system states. This capability helps system administrators in many ways. For example, administrators can understand the evolving behavior of the

[1] For fairness, we have left out our own diagnosis techniques from this paper. Insights from our work could be presented at the workshop.

system, or identify baseline system behavior from the different healthy states of a dynamic system. Furthermore, administrators can prioritize their diagnosis efforts by identifying the frequent failure states and the associated revenue losses.

Task II: Retrieving historic instances from the monitoring data that are similar to one or more given instances. This capability helps system administrators leverage past diagnosis efforts for recurrent failures. Failure diagnosis is expensive in complex systems, so it is valuable to leverage past diagnosis efforts whenever possible; particularly since 50-90% of failures seen are recurrences of previous failures [3]. Furthermore, identifying similar instances from the past helps generate more data that can be input to machine-learning tools. These tools usually give more accurate and confident results when more input (training) data is provided.

Task III: Processing a *diagnosis query Diagnose*(H', F). Here, H' consists of monitoring instances from a healthy system state, while F consists of instances from a failure state. The result of this query is a selected subset of attributes in the monitoring data that indicate the likely cause of the failure represented by F. While Task III is the quintessential task of automated diagnosis, it may need the results of Tasks I and II to generate the inputs H' and F.

3 Techniques Compared for the Three Tasks

We briefly describe the techniques that were implemented and compared for the three tasks above. Please refer to the cited original papers for the full details of each technique.

3.1 Task I: Identifying Distinct System States

The main approach used for solving this task is to divide the historic monitoring data $H \cup U$ into non-overlapping partitions such that instances in each partition have similar characteristics that represent a distinct system state. The frequency of a state can then be estimated as the size of the corresponding partition.

3.1.1 Clustering-Based Techniques

We can partition data using a classical clustering technique like *K-Means*. However, recent work observes that clustering techniques like K-Means suffer from the *curse of dimensionality* in high dimensional spaces [8]. In such spaces, for any pair of instances within the same cluster, it is highly likely that there are some attributes on which the instances are distant from each other. Thus, distance functions that use all input attributes equally can be ineffective. Furthermore, different clusters may exist in different subspaces, i.e., comprised of different subsets of attributes [8].

These issues are addressed by *subspace clustering* (e.g., [1, 8]) which discovers clusters in subspaces defined by different combinations of attributes. The *locally adaptive clustering (LAC)* algorithm from [8] associates each cluster C with a weight vector that reflects the correlation among instances in C. Attributes on which the instances in C are strongly (weakly) correlated receive a large (small) weight, which has the effect of constricting (elongating) distances along those dimensions.

Extension of the Clustering Process

Reference [7] uses K-Means (with K=5) as a building block to produce *pure* clusters iteratively. The purity of a cluster is defined based on the annotation attribute: a cluster is completely pure if it contains failure instances only or it contains healthy instances only. (We will later extend the definition of purity to multiple different failure types.) The extended clustering process in [7] works as follows:

1. Apply K-Means to the historic data and get k clusters.
2. Compute the purity score of each cluster with an *entropy* formula equal to $-p_0 \times log_2(p_0) - p_1 \times log_2(p_1)$, where p_0 and p_1 are the percentage of failures instances and healthy instances in the cluster respectively. A cluster is pure if its entropy is 0.
3. Apply K-Means again to the instances in clusters whose purity scores are below a predefined threshold.
4. Go to Step 2 unless each produced cluster has a purity score below the threshold, or the number of clusters has reached the allowed maximum K_{tot}.

The authors of [7] explore the space of possible values for K_{tot} to compare the clustering techniques.

3.1.2 Classification and Regression Trees (CART)

A *classification tree* (CART, or decision tree) [14] can be learned from the historic data $H \cup U$ with the annotation attribute as the class attribute, and the others as predictor attributes. The tree's leaf nodes represent a partitioning of the data. A nice feature of this approach—which arises from its roots in classification as opposed to clustering—is that the tree minimizes the chances of putting instances from healthy states and instances from failure states into the same partition. We can vary the *pruning level* [14] of the tree during learning to generate different numbers of partitions (leaf nodes) from $H \cup U$. (Pruning is usually used to avoid overfitting the training data.) More aggressive pruning will generate a tree with fewer leaf nodes (partitions).

3.2 Task II: Retrieving Similar Historic Instances

Given an instance f, we can retrieve the top N historic instances from $H \cup U$ that are the most similar to f. The similarity between two instances f and f' can be measured by the inverse of their pairwise Euclidean distance.

The retrieval of f based on a CART learned from historic data will return all instances from the leaf node that the CART classifies f into. The intuition here is that the returned instances have similar patterns in attribute values (i.e., satisfy a similar set of conditions) as f.

3.3 Task III: Processing Diagnosis Queries

Given a diagnosis query $Diagnose(H', F)$, where $H' \subset H$ and F contains instances during or just before a failure, the

task of processing this query is to track down the subset of attributes that pinpoint the cause of the failure represented by F. Previous techniques for processing *Diagnose*(H', F) belong to one of the following two categories:

- The *correlation-based* approach localizes the failure to attributes that correlate highly with the annotation attribute. Examples include [5] and [6] which use CART and *Bayesian network* classifiers [14] respectively to capture correlation. These statistical models are learned from $H' \cup F$ with the annotation attribute used as the class attribute, and the others as predictor attributes.
- The *baselining-based* approach (e.g., [2]) learns the baseline system behavior from the healthy data H'. The failure is localized to attributes whose distribution in the failure data F differ significantly from the baseline.

We now describe the techniques compared for Task III.

3.3.1 Diagnosis using Metric Attribution

The *metric-attribution approach* from Reference [6] processes a $Diagnose(H', F)$ query as follows:

1. Apply feature selection techniques [14] to $H' \cup F$ to remove irrelevant attributes.
2. Learn a *tree-augmented Bayesian network* classifier [14] *TAN* from the filtered $H' \cup F$.
3. For each failure instance $f \in F$, infer which attributes in $X_1, \ldots, X_n$ have values closer to their characteristic values in the failure state than to their characteristic values in the healthy state. This condition can be expressed as $P(m_i | m_{p_i}, A = \text{`failure'}) > P(m_i | m_{p_i}, A = \text{`healthy'})$, where m_i is X_i's value in f, m_{p_i} represents values in f of X_i's parents in *TAN*, A is the annotation attribute, and $P(\cdot|\cdot)$ is a conditional probability distribution given by *TAN*. Intuitively, if X_i's value m_i in the failure instance f is more likely to come from the failure state, then X_i is *attributable* to the failure; and will be output in the diagnosis result.

3.3.2 Diagnosis using CART

The CART-based diagnosis approach [5] works as follows:

1. Learn a classification tree from $H' \cup F$.
2. As a noise-filtering step, remove the leaf nodes in the tree that contain less than a specified threshold of failure instances from F. (This threshold is set to 10% in [5].)
3. Apply Step 2 iteratively until no more leaf nodes can be removed.
4. Output the attributes used in the decision nodes in the leftover tree as the diagnosis result.

3.3.3 Diagnosis using Anomaly Detection

The baselining-based approach described in [2] first captures the distribution D_{X_i} of attribute X_i's ($1 \leq i \leq n$) values in the healthy instances H'. D_{X_i} is approximated by a Gaussian distribution in [2]. Then, [2] computes the probability of D_{X_i} having produced the measurements of X_i in the failure instances in F. If this probability is lower than a threshold, then X_i is included in the diagnosis result.

3.4 Constructing Signatures from Monitoring Data

The techniques discussed so far for the three tasks work directly on the raw monitoring data. Reference [7] applies metric attribution (Section 3.3.1) to construct a *signature* for each instance in the historic data. (This approach is one example of applying *transformations* to the raw data.) The signature of an instance aims to distill the system state that the instance belongs to. Partitioning and retrieval can be done with the constructed signatures instead of the raw data, with the hope that the quality of results for Tasks I and II will be improved. Signatures are constructed as follows:

1. Divide the historic data $H \cup U$ into non-overlapping consecutive chunks C_i based on a preset chunk size. Each chunk contains a minimum number of healthy instances and a minimum number of failure instances.
2. For each chunk C_i, apply feature selection to remove irrelevant attributes; and learn a Tree-augmented Bayesian network *TAN*$_i$ from C_i.
3. A signature is generated for each instance (healthy or failure) in chunk C_i. The signature contains a value for each attribute: if the attribute is removed during the feature selection process, then its corresponding value in the signature is 0; if the attribute is attributable to the failure state, then its value is 1; otherwise its value is -1.

4 Experimental Evaluation

We evaluate the techniques described in this paper in the context of common failures in a three-tier Web service. We implemented a testbed that runs *Rubis* [13]—a multitier auction service modeled after eBay—on a JBoss application server (with an embedded Web server) and a MySQL DBMS. It has been reported that software problems and operator errors are the common causes of failure in Web services [12]. We inject such failures into a running Rubis instance using a comprehensive failure-injection tool [4]. This setting makes it easy to study the accuracy of various diagnosis algorithms because we always know the true cause of each failure.

Specifically, we can inject 3 independent causes of failure—software bugs, data corruption, and uncaught Java exceptions—into 25 Java modules (*Enterprise Java Beans (EJBs)*) that comprise the part of Rubis running in the application server. Using this mechanism, we can inject 75 distinct single-EJB failures and any number of independent multiple-EJB failures (concurrent single-EJB failures).

We collect data for each failure type *ft* as follows: (1) run the testbed under a stable workload (there is no performance bottleneck for this workload) for 20 minutes; (2) inject the specific failure *ft* into the testbed and continue the run for 20 minutes; (3) fix the failure by *microrebooting* [4] affected EJBs, and go to Step 2. The whole process lasts 200 minutes. The monitoring data used in the evaluation consists of the number of times each distinct EJB procedure call is invoked per minute; consisting of a total of 110

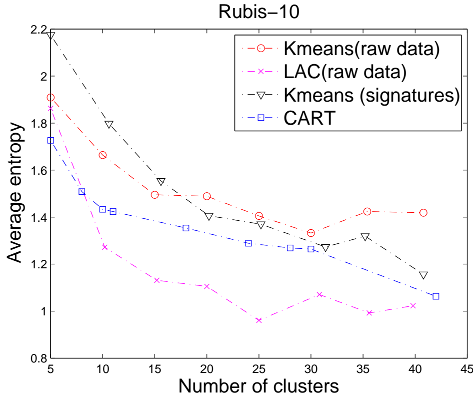

Figure 1: Average entropy for Rubis-10

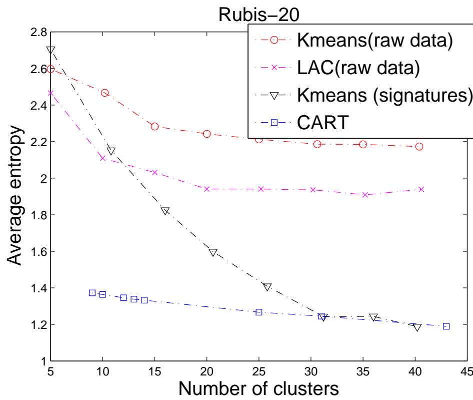

Figure 2: Average entropy for Rubis-20

attributes. (Monitoring at this level is light-weight.) SLO violations are defined based on Rubis's average response time per-minute. For evaluation purposes, we assign an *annotation* to each data instance being collected as follows: (i) healthy, if the instance has no SLO violation, or (ii) the identifier of the injected failure type, if the instance has SLO violations. A failure type is defined by which causes of failure were injected into which EJBs.

The *Rubis-5*, *Rubis-10*, and *Rubis-20* monitoring datasets in Table 1 are used in the evaluation. All experiments were run on a machine with 2 GHz CPU and 1 GB memory.

4.1 Evaluation of Task I

The entropy formula from Section 3.1.1 can be generalized to multiple annotations as $\sum_i -p_i \times log_2(p_i)$, where p_i is the percentage of instances with a specific annotation in a cluster. The quality of a partitioning result can be measured as the average entropy over all produced clusters weighted by their normalized cluster sizes. Generally, the lower the average entropy, the better the partitioning. An average entropy close to 0 means that each cluster mostly contains instances from one system state; so the partitioning characterizes different system states correctly.

We compare four partitioning techniques: (1) K-means applied on the raw monitoring data, (2) LAC applied on the raw monitoring data, (3) K-means applied on signatures constructed from the monitoring data, and (4) CART. Recall from Section 3.4 that we need to set the chunk size for signature construction. We set the chunk size to 100 to ensure that each chunk has sufficient healthy and failure instances to learn a reasonable Bayesian network. We do not have a direct control over the number of partitions generated by CART. We can only vary the pruning level that affects how many leaf nodes (partitions) remain in the produced tree.

Name	a	i	Data and injected failures
Rubis-5	110	986	From 5 distinct single-EJB failures
Rubis-10	110	1971	From 10 distinct single-EJB failures
Rubis-20	110	3972	From 20 distinct single-EJB failures

Table 1: Monitoring datasets used. Columns a and i are the number of attributes and instances respectively.

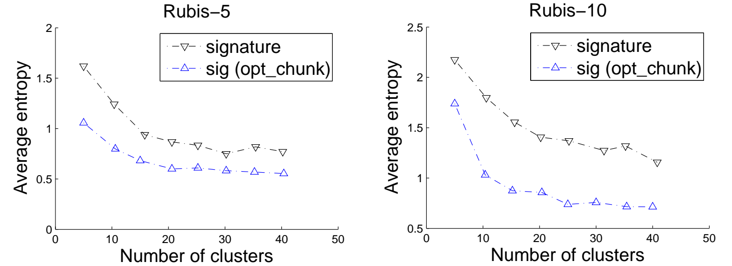

Figure 3: Average entropy: (a) Rubis-5, (b) Rubis-10

Figures 1 and 2 show the average entropy of each partitioning technique as we increase the number of maximum allowed partitions for Rubis-10 and Rubis-20 respectively. We observe the following trends:

- As the number of partitions increases, the average entropy generally decreases since the produced partitions become more pure (as expected).
- LAC performs better than K-means. As the dimensionality of our monitoring data is high (>100), the computation of Euclidean distance in the full space by K-means is problematic. LAC avoids this problem by searching for clusters in lower-dimensional subspaces.
- CART shows relatively good performance because it produces many low-entropy clusters that mostly contain healthy instances. Recall that CART is good at separating healthy instances from failure instances, but not necessarily at separating failure instances of different types from each other. Our monitoring datasets contain roughly equal amounts of healthy and failure instances; giving CART an undue advantage.
- Comparing K-means applied on signatures with K-means on the raw monitoring data gives mixed results. For Rubis-10, both show similar performance. For Rubis-20, K-means on signatures is much better. It turns out that the quality of signatures is very sensitive to the chunk size used during signature construction (Section 3.4). If the chunk size is too big, then each chunk may include instances from different system states. If the chunk size is too small, then each chunk may not contain enough instances to learn a reasonable Bayesian network. In both cases, metric attribution may not function as desired, leading to incorrect signatures.

To verify our observations about signatures, we analyzed the possible improvement when the chunk size is set

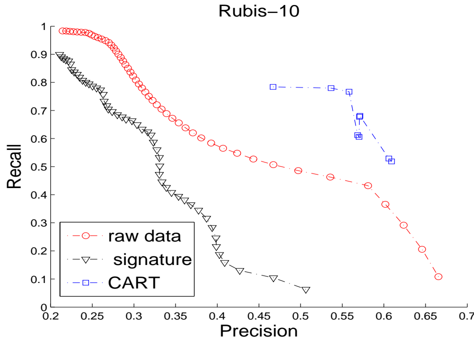

Figure 4: Precision-recall curves for Rubis-10

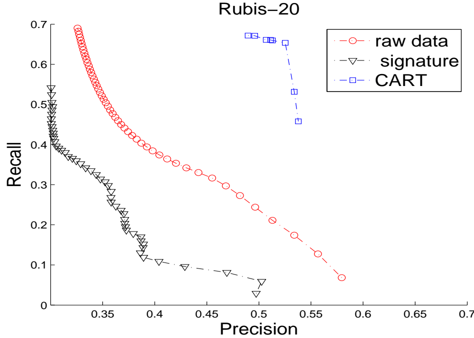

Figure 5: Precision-recall curves for Rubis-20

optimally (which is possible only when we know the points where system resources, workload, or configuration changed). Figure 3 plots the average entropy of K-means clustering on signatures for Rubis-5 and Rubis-10 respectively in two settings: (1) the chunk size is fixed at 100 during signature construction (denoted as *signature*); and (2) the chunk size is set adaptively to capture all change points in system state (denoted as *sig(opt_chunk)*). Note the considerable improvement in the quality of partitioning with optimal chunk sizes. The relative improvement also increases as the number of failure types increase.

4.2 Evaluation of Task II

We adopt the traditional metrics of *precision* and *recall* for evaluating retrieval quality. For a given instance f and number N of instances similar to f to be retrieved: precision measures the percentage of correctly-retrieved instances among the N results; and recall measures the percentage of correctly-retrieved instances among all historic instances with the same annotation as f. The perfect value for both metrics is 100%. Intuitively, as N increases, recall improves while precision drops. Thus, the bigger the area under the precision-recall curve, the better.

Figures 4 and 5 plot the precision-recall curves for Rubis-10 and Rubis-20 respectively. N is varied from 20 to 1000. The precision and recall values are averaged over a test set comprising 40% of the data. We could not plot the precision-recall curve for CART in the high-recall and high-precision regions since we do not have direct control on the number of instances retrieved from a CART (recall Section 3.2).

Reference [7] observed that signature-based retrieval is

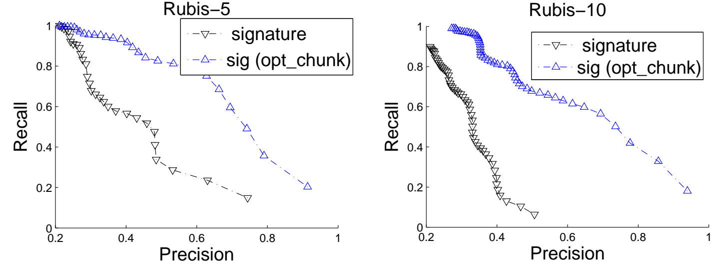

Figure 6: Precision-recall curves: (a) Rubis-5, (b) Rubis-10

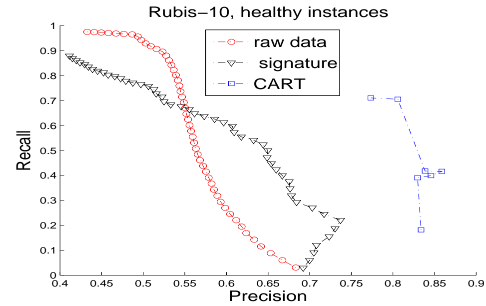

Figure 7: Precision-recall curves for retrieving healthy instances in Rubis-10

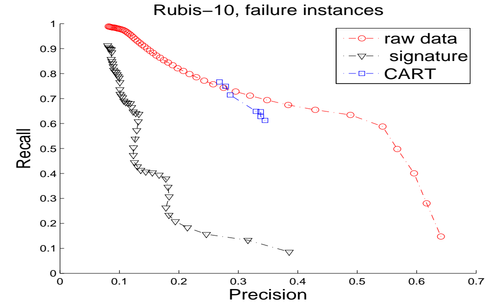

Figure 8: Precision-recall curves for retrieving failure instances in Rubis-10

better than raw-data-based retrieval. However, Figures 4 and 5 show contrary results. Once again, the possible reason is that signature construction is sensitive to the chunk-size setting; but there is no principled way to set this value statically. This point is verified in Figure 6 which shows the precision-recall curves for Rubis-5 and Rubis-10 for the fixed chunk-size and optimal chunk-size settings (recall Section 4.1). There is considerable improvement in both precision and recall with optimal chunk sizes.

Retrieval based on CART performs better than the other two techniques because CART retrieves healthy instances with higher precision than the other two. This point is verified in Figures 7 and 8 which respectively plot the precision-recall curves for retrieving healthy instances only and failure instances only in Rubis-10.

4.3 Evaluation of Task III

Since we know the actual root cause of each failure injected in the testbed, we can evaluate the accuracy of a diagnosis technique in terms of the *false positives (FP)* and *false negatives (FN)* in its result. FP counts the number of EJBs that appear in the diagnosis result but are not the root cause of failure. FN counts the number of EJBs that cause the fail-

Failure/(FN, FP)	MA	CART	AD
Exception-Bid	0, 7	0, 3	0, 2
Exception-BuyNow	0, 1	0, 2	1, 2
Exception-Category	0, 2	0, 0	1, 3
Exception-SB_AboutMe	1, 2	0, 4	1, 3
Exception-SB_Auth	1, 3	1, 2	1, 3
jndi-Bid	0, 3	0, 0	0, 4
jndi-BuyNow	0, 5	1, 1	1, 3
jndi-Category	0, 4	0, 0	0, 3
jndi-SB_AboutMe	0, 5	1, 1	1, 3
jndi-SB_Auth	1, 10	1, 1	1, 2
nullmap-Bid	0, 2	0, 0	0, 2
nullmap-BuyNow	0, 5	0, 4	1, 2
nullmap-Category	0, 12	0, 0	1, 1
nullmap-SB_AboutMe	0, 1	0, 4	1, 2
nullmap-SB_Auth	1, 3	1, 1	1, 2

Table 2: Comparing diagnosis accuracy (see Section 3.3); each entry gives false negatives (FN), false positives (FP)

ure but are missing from the diagnosis result. Both false positives and false negatives hurt diagnosis accuracy.

Table 2 compares the diagnosis accuracy of the three techniques: metric attribution (MA), CART, and anomaly detection (AD). The failures were injected one at a time, and come from a combination of ten distinct EJBs and three distinct failure types: Java exceptions (Exception), data corruptions (jndi), and software bugs (nullmap). (Table 2 shows a subset of the results.) MA performs slightly better than the other two techniques in capturing the root cause. MA fails 10 out of 30 times to catch the true cause (FN = 1). However, MA tends to produce more false positives than CART. A good combination of these techniques may reduce both false positives and false negatives. A manual analysis of the results showed that FP can be reduced by incorporating domain knowledge—e.g., inter-EJB calling patterns which lead to fault propagation [4]—alongside the statistical machine-learning analysis.

5 Summary

We empirically compared techniques for three tasks related to diagnosis of system failures. While a number of empirical insights were generated, there was no dominant winner among the techniques we compared. Our evaluation showed plenty of room for improving these techniques in the following aspects to realize industrial-strength automation of failure diagnosis, and to ultimately make systems detect, diagnose, and fix failures automatically (*self-healing*):

- Most of the basic techniques involved in our comparisons (clustering, CART, Bayesian networks) regard each instance in the data as independent of others, disregarding strong temporal correlations that may exist. This observation poses both challenges and opportunities. For example, techniques that can detect transition points between distinct system states can be combined with signature construction to improve diagnosis results significantly (recall Section 4).
- Failure propagation [4] exposes some brittleness in the pure machine-learning-based techniques that we compared. Note that most rows in Table 2 have at least one technique that catches the true cause (i.e., FN = 0). Hence, some *symptom* that maps directly to the true cause is indeed present in the data in most cases. (Failures injected in the SB_Auth EJB are the interesting exceptions.) However, this symptom may be overlooked, possibly because it is not the dominant one in the data. Addressing this issue—e.g., by combinating the current techniques or incorporating domain knowledge through *symptom databases* [7]—is an important next step.
- Monitoring data (especially failure data) from production systems can contain various types of noise (recall Section 1). Making diagnosis techniques robust to such noise is a promising direction for future work.

6 Acknowledgements

We are extremely grateful to Carlotta Domeniconi for providing us the source code for LAC, and to George Candea and other members of Stanford's Software Infrastructure Research Group for helpful discussions and source code.

References

[1] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park. Fast algorithms for projected clustering. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 1999.

[2] P. Bodik, G. Friedman, L. Biewald, H. Levine, G. Candea, K. Patel, G. Tolle, J. Hui, A. Fox, M. I. Jordan, and D. Patterson. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In *Proc. of IEEE Intl. Conference on Autonomic Computing*, June 2005.

[3] 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 *Proc. of 2nd IEEE International Conference on Autonomic Computing (ICAC)*, 2005.

[4] G. Candea, M. Delgado, M. Chen, and A. Fox. Automatic failure-path inference: A generic introspection technique for internet applications. In *Proc. of IEEE Workshop on Internet Applications (WIAPP)*, 2003.

[5] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *Proc. of the 1st IEEE International Conference on Autonomic Computing (ICAC)*, 2004.

[6] 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.

[7] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, and T. Kelly. Capturing, indexing, clustering, and retrieving system history. In *SOSP*, 2005.

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

[9] S. Duan and S. Babu. Processing forecasting queries. In *Proc. of the 2007 Intl. Conf. on Very Large Data Bases*, 2007.

[10] S. Duan and S. Babu. Guided problem diagnosis through active learning. In *Proc. of 5th IEEE International Conference on Autonomic Computing (ICAC)*, June 2008.

[11] S. Duan and S. Babu. Fa: A system for automating failure diagnosis. In *Proc. of the 2009 Intl. Conf. on Data Engineering*, 2009.

[12] 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.

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

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