ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-569

A Framework for Automatic Debugging of Functional and Degradation Failures

Nuno Cardoso[1] and **Rui Abreu**[2] and **Alexander Feldman**[3] and **Johan de Kleer**[4]

Abstract. Software diagnosis is a particularly challenging problem for modern systems, which may consist of dozens, if not hundreds, of components computing on concurrent and potentially distributed platforms, and using infrastructure and services built by many organizations. We propose a framework that generalizes state-of-the-art classical reasoning-based fault diagnosis which tolerates observation uncertainty and addresses degradation of quality of service. Empirical evaluation involving 27 000 highly realistic synthetic scenarios demonstrates an average accuracy improvement of 20% (with 99% statistical significance) which is considerable in the domain of Software Fault Localization (SFL). We measure the improvement in accuracy on well-established SFL performance metrics.

Introduction

One of the most important way to improve the trustworthiness of software systems is to increase their robustness in the face of (run-time) failures. While design-time methods are useful in improving confidence in software (e.g., [3, 13, 16, 19, 22, 31]), they cannot by themselves eliminate the possibility of run-time failures, which are induced by a variety of factors largely outside the control of the organization producing that software: faults in runtime infrastructure and components provided by third-parties, unpredictable loads, variable resources, and malicious attempts to break a system. Moreover, as mentioned in [12], the distinction between "healthy" and "broken" is often indistinct and fuzzy, and there is a gradual transition, over time, between these two states [12]. Consequently, stakeholders must take increasing responsibility for improving the trustworthiness of their systems through building automatic runtime problem detection and repair [12, 23].

Diagnosis for today's complex systems, however, is particularly challenging. First, the presence of concurrency makes it difficult to identify which computation might have caused a problem. Second, reliance on middleware for distributed communication, and more generally the use of components and infrastructure produced by many organizations, means that in many cases neither specifications nor code is available for all parts of the system. Third, in many systems, problems may be intermittent, caused by transient faults or variability in loads. Fourth, many of the "faults" that we care about are reflected indirectly by violation of a systems quality of service goals, such as degradation of response latency, rather than by a direct failure such as a server or system crash. Such "soft" faults may be difficult to detect and diagnose [12]. Consequently, although fault diagnosis has been studied extensively for both hardware and software systems as a development time activity, the ability to do this at run-time (i.e., while the system is operational) in a systematic way for complex systems has remained an elusive goal [8].

As no behavioral models are typically available, current approaches to software diagosis abstract the system under analysis in terms of component activity and correct/incorrect behavior, notably lacking mechanisms to encode soft faults.

We propose a framework that generalizes state-of-the-art classical reasoning-based fault diagnosis (such as, Spectrum-based Fault Localization (SFL) [4], GDE [19]) to accommodate functional and degradatation failures. In particular, the framework is capable of reasoning under uncertainty (there is a variety of sources of uncertainty as the ability to observe the behavior of a system may be limited by the kinds of monitoring infrastructure available) and handle soft faults. In many cases the existence of a fault is linked to degradation of quality of service. For example, high latencies of responses to queries may indicate that servers are overloaded, that a network connection is faulty, or both.

Our framework improves the classical reasoning approach in 65% of the cases and achieved at least equal performance in 94% of the cases. The overall relative improvement in the diagnostic quality was of 20% on average, with a 99% confidence interval.

This paper makes the following contributions:

- We discuss the limitations imposed by the classical reasoning-based fault diagnosis;
- We propose a generalization of the classical reasoning-based diagnostic framework aimed at improving its accuracy when diagnosing soft faults;
- We compare the accuracies of the classical and our novel approach using a simulation-based setup, which has been shown to be able to generate realistic scenarios [9].

Reasoning-Based Diagnosis

In this section we introduce concepts and definitions used throughout the paper, as well as the reasoning-based SFL approach to diagnosis.

Definition 1 (Diagnostic System)**.** *A diagnostic system DS is a set of components $COMPS = \{c_1, c_2, \ldots, c_m\}$.*

The type of systems we consider typically consists of hundreds to thousands of components. These components can be code-blocks, assembly-level instructions or whole state machines. The systems can be distributed, hybrid, and can contain network components such as routers and balancers.

[1] University of Porto and HASLab / INESC TEC, Portugal. email: nunopcardoso@gmail.com
[2] Palo Alto Research Center, Inc, USA. email: rui@parc.com
[3] Palo Alto Research Center, Inc, USA. email: afeldman@parc.com
[4] Palo Alto Research Center, Inc, USA. email: dekleer@parc.com

Definition 2 (Transaction)**.** *A transaction* $\langle A, e\rangle$ *is a pair containing* $A \subseteq COMPS$ *and the transaction outcome* $e \in \{0, 1\}$.

Transactions are typically computed by executing programs or program tasks and recording success or failure e. The program components that participate in A are instrumented by using debugger-like methods [14]. The convention is that $e = 1$ means failure and $e = 0$ means success. Transactions where $e = 1$ are also known as conflicts [19]. A conflict represents a set of components that cannot be simultaneously healthy to explain the observed erroneous behavior.

Definition 3 (Hit Spectrum)**.** *A hit spectrum* $\mathbf{A}$ *is a set of transactions* $\mathbf{A} = \{\langle A_1, e_1\rangle, \langle A_2, e_2\rangle, \cdots, \langle A_n, e_n\rangle\}$.

We assume that an external diagnostic engine [21, 28, 11, 1, 5] computes a set of diagnoses. These diagnoses are used as an input to our algorithm.

Definition 4 (Diagnosis)**.** *A diagnosis* $\langle d, Pr(d)\rangle$ *is defined by a set of components* $d \subseteq COMPS$ *and a prior probability* $Pr(d)$.

The prior probability $\Pr(d)$ estimates to what extent a candidate, without further evidence, is responsible for the system's malfunction. To define $\Pr(d)$, let p_j denote the prior probability that a component c_j is at fault. The value of p_j is application dependent. In the context of development-time fault localization, p_j is often approximated as $p_j = 1/1000$, i.e., 1 fault for each 1000 lines of code [7].

Assuming that components fail independently, the prior probability for a particular diagnosis d is given by

$$\Pr(d) = \prod_{j \in d} p_j \cdot \prod_{j \in COMPS \setminus d} (1 - p_j) \tag{1}$$

When the p_j are equal the larger the candidate the smaller its *a priori* probability will be.

This leads us to our main goal which is to apply a Bayes conditioning rule.

Definition 5 (Bayes Conditioning)**.** *Given a diagnosis* d *and a hit spectrum* $\mathbf{A}$*, the a posteriori probability* $Pr(d|\mathbf{A})$ *is:*

$$Pr(d|\mathbf{A}) = Pr(d) \cdot \prod_{i \in 1,2,\ldots,N} \frac{Pr(A_i, e_i \mid d)}{Pr(A_i)} \tag{2}$$

In order to characterize the optimality of our algorithm we need to compute posteriori probability (given $\mathbf{A}$) for a whole set of diagnoses and to rank them. This gives us our main computational problem:

Problem 1 (Diagnostic Ordering)**.** *Given a set of diagnoses* $D = \{\langle d_k, Pr(d_k)\rangle\}$ *for* $k \in \{1, 2, \ldots, K\}$ *and a hit spectrum* $\mathbf{A}$*, compute* $Pr(d_k|\mathbf{A})$ *and order* D *such that:*

$$\forall d_k \in D : Pr(d_k|\mathbf{A}) \geq Pr(d_{k+1}|\mathbf{A}) \tag{3}$$

In what follows we describe the relevant aspects of the classical reasoning-based SFL approach to address the ranking problem [3, 19].

To simplify computation we assume conditional independence throughout the process, i.e., our Bayes classifier is naïve.

The denominator $\Pr(A_i)$ is a normalizing term that is identical for all $d \in D$ and needs not to be calculated for ranking purposes as it does not alter the rank order.

To bias the prior probability taking run-time information (i.e., observations) into account, $\Pr(A_i, e_i \mid d)$ (referred to as likelihood) is defined as

$$\Pr(A_i, e_i \mid d) = \begin{cases} \mathrm{G}(d, A_i) & \text{if } e_i = 0 \\ 1 - \mathrm{G}(d, A_i) & \text{otherwise} \end{cases} \tag{4}$$

$\mathrm{G}(d, A_i)$ (referred to as transaction goodness) is used to account for the fact that components may fail intermittently, estimating the probability of nominal system behavior under an activation pattern A_i and a diagnostic candidate d.

Let g_j (referred to as component goodness) denote the probability that a component c_j performs nominally. Considering that all components must perform nominally to observe a nominal system behavior, $\mathrm{G}(d, A_i)$ is defined as

$$\mathrm{G}(d, A_i) = \prod_{j \in (d \cap A_i)} g_j \tag{5}$$

In scenarios where the values for g_j are not otherwise available, those values can be estimated by maximizing $\Pr(A, e \mid d)$ (Maximum Likelihood Estimation (MLE) for naïve Bayes classifier) under parameters $\{g_j \mid j \in d \wedge 0 \leq g_j \leq 1\}$ [3].

Approach

In this section we discuss how degradation failures can be more accurately detected/represented and how the diagnostic framework presented in the previous section can be enhanced to more accurately diagnose such kind of errors.

Fuzzy Error Detection

The first challenge in diagnosing soft failures related to their detection. Existent approaches to error detection (e.g., [8], SFL [4], and GDE [19]) make use of first-order logic descriptions of the correct behavior of the system (weak-fault models) to assign transactions to one of two possible sets: the pass set and the fail set (P and F respectively, where $F = \overline{P}$). A consequence of such fault models is the *crisp* distinction between correct and incorrect system states. While this crisp logic description enables an accurate representation of hard failures, it is unable to accurately represent a large variety of soft failures. Take for instance a type of soft failure that, informally, can be described by the statement "The system is slow." Even though we can easily relate the slowness of the system to an appropriate metric (e.g., response time), it is not easy to define a crisp boundary in this same metric to distinguish acceptable and slow transactions. By setting a crisp boundary at, for instance, 1 second, a response time of 0.9999 seconds would be considered to be correct whereas a marginally superior response time would be considered incorrect. Also, a response time of 0.9999 seconds would result in the same type of error information (pass) as a smaller response time even though the larger response time may represent an error symptom.

To overcome the expressiveness limitation of crisp logic error detection mechanisms, we propose a generalization using fuzzy logic [29]. Fuzzy logic extends the notion of binary set membership by introducing the concept of membership functions, denoted μ_ω (membership function for set ω – in the context of this paper $\omega \in \{F, P\}$), that map a particular domain on the real continuous interval $[0, 1]$, where the endpoints of 0 and 1 conform to no membership and full membership, respectively. Let x be an arbitary event. In the context of error detection, the concept of fuzzy membership enables the representation of 3 types of system states:

correct: $\mu_F(x) = 0$,
incorrect: $\mu_F(x) = 1$, and
degraded: $0 < \mu_F(x) < 1$

As a consequence of the new fuzzy error model, a degraded transaction exhibits both correct and incorrect behaviors simultaneously, however with different degrees. As $F = \overline{P}$, it follows that

$$\mu_P(x) = 1 - \mu_F(x) \tag{6}$$

As an example, consider the crisp fail set containing all response times (rt) above 1 second. This same set could be represented in terms of a membership function as

$$e_{\text{crisp}}(rt) = \mu_F(rt) = \begin{cases} 0 & , \ rt \leq 1 \\ 1 & , \ rt > 1 \end{cases} \tag{7}$$

To achieve the goal of representing soft failures (and implicitly the degraded state), consider that all response times below 0.5 seconds could be considered correct and all times above 1 second incorrect. Furthermore, consider that the amount of degradation follows a linear pattern between those two thresholds. The fuzzy fail set representing this particular type of error could be defined as

$$e_{\text{fuzzy}}(rt) = \mu_F(rt) = \begin{cases} 0 & , \ rt < 0.5 \\ 2 \cdot rt - 1 & , \ 0.5 \leq rt \leq 1 \\ 1 & , \ rt > 1 \end{cases} \tag{8}$$

Both membership functions are presented in Figure 1. It is important to point out that the assumption of linear degradation introduced in Equation 8 was only for simplicity. In real-world scenarios, the membership functions are application dependent and can exhibit arbitrary patterns. We treat the membership functions are treated as black-boxes. Figure 2 shows a set of alternative membership functions for the error previously described. Despite their odd shapes they are acceptable membership functions, provided that they correctly describe the error state of the transaction.

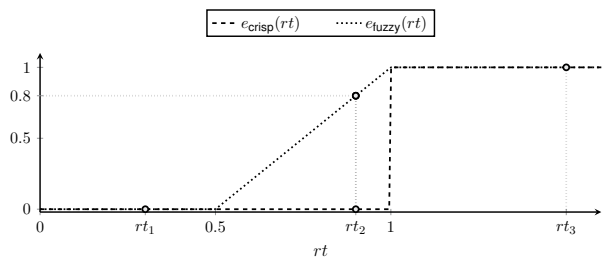


Figure 1: Crisp vs. fuzzy sets

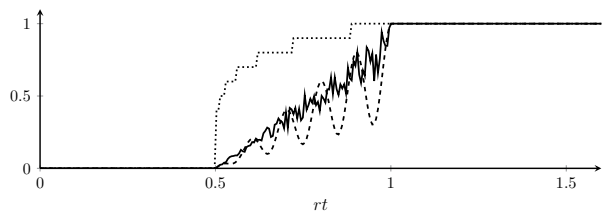


Figure 2: Arbitrary membership functions

Furthermore, it is possible to change the error detection sensitivity by raising e_{fuzzy} to an exponent, as depicted in Figure 3 using μ_F. We can see that by raising e_{fuzzy} to an exponent in the interval $[1, +\infty]$, the error detection becomes less sensitive to errors in the *fuzzy* zone (i.e., the error value in the fuzzy zone becomes smaller than the original). In contrast, by raising e_{fuzzy} to an exponent in the interval $[0, 1]$, the error detection becomes more sensitive to errors. In fact, when the exponent tends to either $+\infty$ or 0, the fuzzy membership becomes a binary membership and errors in the fuzzy zone become passes and fails, respectively.

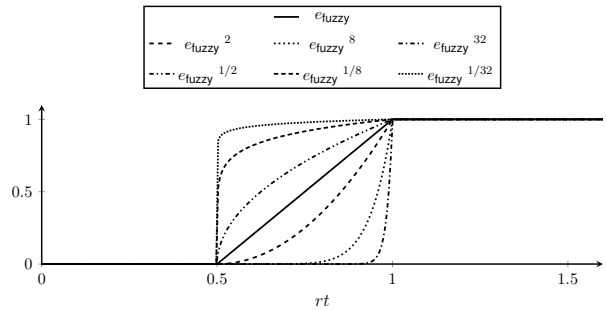


Figure 3: Error detection sensitivity intuition

t	rt	A	e	
			$e_{\text{crisp}}(rt)$	$e_{\text{fuzzy}}(rt)$
1	0.3	$\{c_2\}$	0	0
2	0.9	$\{c_1\}$	0	0.8
3	1.5	$\{c_1, c_2\}$	1	1

Table 1: Fuzzy error hit spectrum example

To illustrate the fuzzy error detection process, consider the spectrum presented in Table 1, which also contains the run-times for each transaction (marked in Figure 1). From this spectrum we can see that, in particular for t_2, the crisp error vector neglected an error symptom whereas the fuzzy error vector categorized that same transaction as being 80% degraded.

Fuzzy Error Diagnosis

Using fuzzy logic to detect errors, it is possible to assert that a particular transaction is 80% degraded (i.e., $\mu_{\widetilde{F}} = 0.8$ and consequently $\mu_{\widetilde{P}} = 0.2$). The remaining challenge consists in integrating this additional knowledge in the diagnostic process.

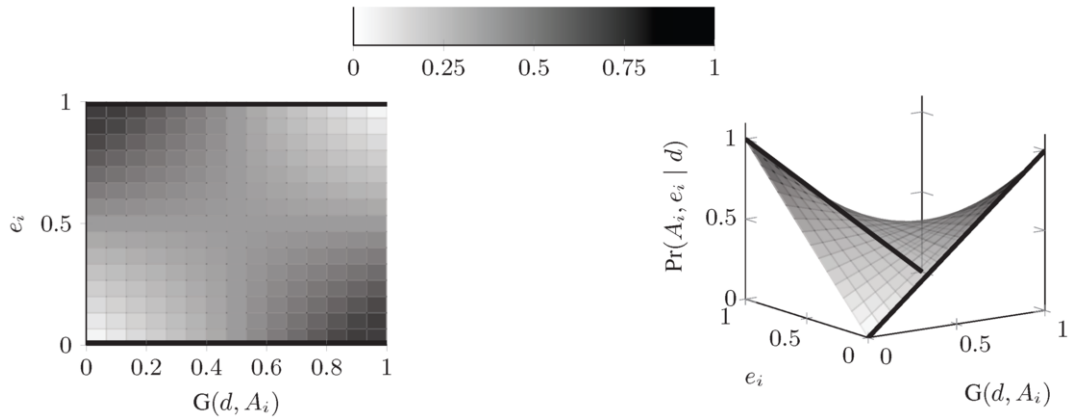
As an example, consider again the spectrum depicted in Table 1. Using the approach explained in the previous section, more concretly Barinel [3], with $e \in \{0, 1\}$, it follows that the diagnosis candidates[5] $d_1 = \{c_1\}$ and $d_2 = \{c_2\}$ are ranked equally. This is because the components in both candidates are involved in the same number of passed and failed transactions. However, intuitively we would expect d_1 to be ranked ahead of d_2 since transaction t_2, in which component c_1 was involved, shows error symptoms whereas t_1 does not.

To solve this limitation we make use of the concept of probability of a fuzzy event in Equation 4. From [30], it follows that the probability of a fuzzy event is defined as

$$\Pr(A_i, e_i|d) = \sum_{\omega \in \Omega} \mu_\omega(x) \cdot \Pr(A_i, e_i|d) \tag{9}$$

where $\Omega \in \{F, P\}$ and x is an arbitary event (an observation; rt in

[5] The candidates for the fuzzy approach are calculated by setting a threshold for e to discretize transactions in terms of pass/fail. In this example we use the threshold $e = 1$.



(a) 2D view (b) 3D view

Figure 4: Likelihood function plot

our working example). From Equation 9, Equation 4 generalizes to

$$\Pr(A_i, e_i \mid d) = \overbrace{e_i \cdot (1 - \mathrm{G}(d, A_i))}^{\omega=F} + \overbrace{(1 - e_i) \cdot \mathrm{G}(d, A_i)}^{\omega=P} \quad (10)$$

where $e_i = \mu_F(\omega)$ and $1 - e_i = \mu_P(\omega)$. The term $(1 - \mathrm{G}(d, A_i))$ accounts for Equation 4's first branch, and $\mathrm{G}(d, A_i)$ for the second one. In contrast to Equation 4, this generalization is valid for fuzzy error values (i.e., e = e_{fuzzy}). Figure 4 shows the plot of the Equation 10 with respect to e_i and $\mathrm{G}(d, A_i)$. For comparison, we also plot Equation 4 with thick black lines.

We next illustrate the framework for our running example. The probabilities of the two candidates (d_1, d_2) are calculated as follows

$$\Pr(d|\mathbf{A}) = \Pr(d) \cdot \prod_{i \in 1,2,\ldots,N} \frac{\Pr(A_i, e_i, rt|d)}{\Pr(A_i)} \quad (11)$$

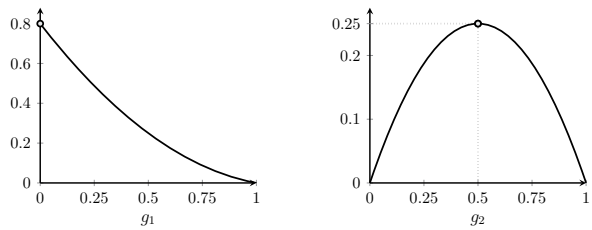
where

$$\Pr(d_1) = \Pr(d_2) = \tfrac{1}{1000} \cdot (1 - \tfrac{1}{1000}) = 9.99 \times 10^{-4} \quad (12)$$

$$\Pr(A, e, rt|d_1) = \underbrace{(0.8 \cdot (1 - g_1) + (1 - 0.8) \cdot g_1)}_{t_2} \times \underbrace{(1 \cdot (1 - g_1) + (1 - 1) \cdot g_1)}_{t_3} \quad (13)$$

$$\Pr(A, e, rt|d_2) = \underbrace{(0 \cdot (1 - g_2) + (1 - 0) \cdot g_2)}_{t_1} \times \underbrace{(1 \cdot (1 - g_2) + (1 - 1) \cdot g_2)}_{t_3} \quad (14)$$

As an example, setting g_1 and g_2 as 0.1 yields the diagnostic ranking $\langle d_1, d_2 \rangle$, where the true faulty explanation is the first place of the ranking. However, as component goodnesses are assumed not to be available, g_1 and g_2 are estimated using maximum likelihood estimation in both symbolic expressions: $\Pr(A, e \mid d_1)$ and $\Pr(A, e \mid d_2)$.



(a) $\Pr(A, e, rt|d_1)$ (b) $\Pr(A, e, rt|d_2)$

Figure 5: Likelihood plots

As can be seen directly from Figure 5, the Prs are maximized when $g_1 \approx 0$ and $g_2 = 0.5$.

Applying the maximizing values to both expressions, it follows that

$$\Pr(A, e, rt|d_1) = 0.8$$

$$\Pr(A, e, rt|d_2) = 0.25$$

.

Hence,

$$\Pr(d_1|\mathbf{A}) = 0.76$$

$$\Pr(d_2|\mathbf{A}) = 0.24$$

The diagnostic ranking is therefore $\langle d_1, d_2 \rangle$, which breaks the ambiguity between d_1 and d_2, thus improving the diagnostic accuracy.

Benchmark

In this section we describe our benchmark approach and discuss results.

Simulator

Performing benchmarks on real applications requires extensive adaptation and is therefore very time consuming. Furthermore, the use of a limited set of applications limits the generalization of the conclusions.

To overcome such issues, we make use of a simulator as proposed in [9][6]. The simulator provides functions to describe and execute a probabilistic model of an arbitrary system, thereby gathering the required spectra. The authors of the simulator showed [9] that the benchmark results for both real and synthetic data are comparable. This is mainly due to the fact that, since systems are highly abstracted, the spectra generated by real and simulated systems is similar.

The simulator consists of a stack automaton which takes as input a probabilistic model of the system (such as, for instance, the one depicted in Figure 6) and, through a Monte Carlo process, generates spectra. The probabilistic model describes the systems topology, the interaction between components, and the systems faults. To build the topology portion of the model, two primitives exist: components and links (depicted in white and light gray, respectively).

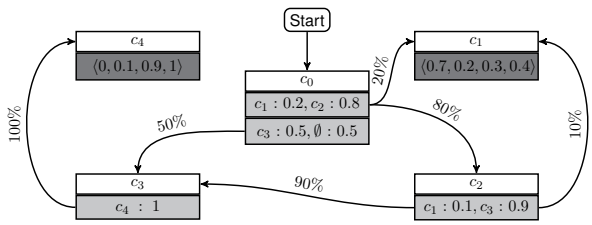


Figure 6: Probabilistic topology model

Concretely, a component (identified by its numeric ID) contains a list of links. A link consists of a list of component IDs with associated transition probabilities ($\emptyset$ corresponds to no transition). Whenever a component is activated all the links belonging to that particular component are sequentially activated. A link is an abstraction to the components' interaction that contains a set of component IDs with their respective call probabilities. With the activation of a link, the current component and link list position are pushed onto a call stack and a component is randomly selected to continue the execution. At the end of the component's execution, an element is popped from the call stack, returning the control to the caller component. Using this model, a transaction can be generated by pushing a component marked as an entry point onto the call stack.

To emulate the error behavior, components may be injected with faults (depicted in dark gray), which are parameterized over 4 variables (p_c, p_d, p_i, and p_f). p_c, p_d, and p_i correspond to the probabilities of correct, degraded, and incorrect behavior. During the simulation, whenever a faulty component is activated, the outcome of such activation (in terms of correct, degraded, or incorrect) is randomly determined using such probabilities. In the event of a component performing erroneously, it has an associated probability p_f of failure which, whenever it occurs, it results in a premature end of the transaction (in Figure 6, an error in component c_4 always results in a failure whereas an error in component c_1 only has a 40% chance of resulting in a failure). To determine the transaction's fuzzy error value, we apply the following rules:

$$e = \begin{cases} 1, & \text{if at least one component performed erroneously} \\ 0, & \text{if all components performed correctly} \\ \text{rand}(0,1), & \text{otherwise} \end{cases} \tag{15}$$

[6] https://github.com/SERG-Delft/sfl-simulator

Setup

We generate the spectra required for our benchmark in two phases. In the first stage, we randomly generate a set of system models, while in the second, we use such models to generate the required spectra.

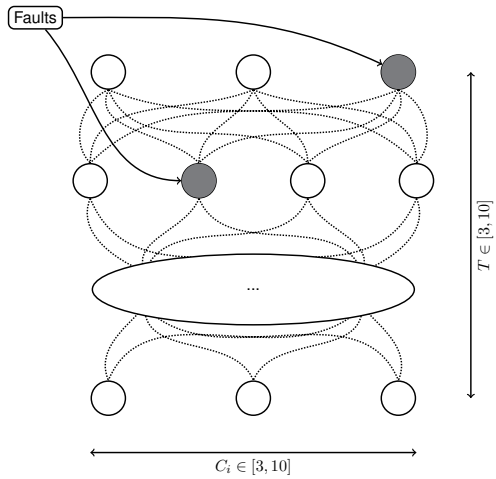


Figure 7: N-tier service architecture

We generated system models that comply with a N-tier service architecture (Figure 7). Systems were created by randomly selecting the number of tiers ($T \in [3, 10]$) as well as the number of components in each tier ($C_i \in [3, 10], 1 \leq i \leq T$). Every component is connected to all the components of the next tier with random transition probabilities. To exhibit erroneous behavior, a number of faults (F) was randomly injected (in terms of position) in the systems.

In our setup, we generated 100 systems for each value $F \in [2, 4]$, totaling 300 systems. The injected faults had 90% and 10% probabilities of degraded, and erroneous behavior, respectively.

The spectra generation is parameterized over a single variable E, representing the number of errors at the end of which the simulation stops. For each generated system, we ran 10 simulations for each value $E \in [1, 9]$, totaling 90 spectra per system. Overall, our benchmark is composed of $300 \times 90 = 27000$ test cases.

Metrics

The wasted effort metric evaluates how many components need to be inspected before all faulty components are found [25]. To calculate this metric one must undergo an iterative process. Starting with the first candidate, all of the candidate's components are inspected to determine whether or not that particular set of components was responsible for the erroneous behavior. Depending on the result of such inspection two outcomes may occur. On the one hand, if the component is found to be faulty, that particular component is removed from all other candidates in the ranking. On the other hand, if the component is found to be healthy, all candidates in the ranking containing that particular component are removed. This process is repeated until all faulty components are found. In the case of the last inspected candidate being tied with other candidates, it is assumed that, on average, half of the healthy components are examined.

During this iterative process, we keep track of two counters: inspected components (I) and faulty components (C). Using these two counters, the wasted effort metric is calculated as

$$W = I - C \tag{16}$$

	d	Rank	I	C
1	$\{c_1, c_4\}$	1	2	1
2	~~$\{c_2, c_3, c_4\}$~~	2		
3	~~$\{c_3, c_4, c_5\}$~~	3		
4	$\{c_1, c_2\}$	4	4	2
5	$\{c_3, c_5\}$	4		

Table 2: Example diagnostic report

As an example consider the diagnostic report presented in Table 2 for which the correct diagnostic candidate is $d = \{c_1, c_2\}$. In order to calculate the wasted effort, we start by examining c_1 and c_4 finding that c_1 is faulty while c_4 is healthy. Due to c_4 being healthy, candidates d_2 and d_3 are not examined. Examining d_4 we observe that the only unexplored component (c_2) is faulty. Additionally, we see that both system's faults were discovered. However, as d_5 is tied with d_4, we must inspect half of the healthy components. The wasted effort of this diagnosis is therefore $W = 4 - 2 = 2$, meaning that 2 healthy components (c_4 and c_3/c_5) were examined in the process of finding the root cause of the system errors.

A normalized version of the wasted effort is called diagnostic quality and is defined as

$$Q = 1 - W/(M - C) \tag{17}$$

where M is the number of system components. The diagnostic quality value is contained between zero and one and estimates the fraction of system's healthy components that need to be examined before all faulty components are found.

In this paper we refine the diagnostic quality metric to take into account the fact that, for a specific spectrum, not all components of the system can be at fault. As an example consider a system with 1000 components with a spectrum consisting of a single failing transaction activating 2 components. Assuming the diagnostic algorithm only proposes plausible[7] candidates, the quality is contained in the interval between 1 and $\frac{999}{1000}$. Instead of calculating the diagnostic quality using the M components of the system, we use M_s, the number of "suspicious" components to calculate the new metric, which shall be referred to as "fair quality" (Q_f). A component is said to be suspicious if it was activated in a failing transaction. A consequence of using Q_f is that the diagnostic qualities of all possible permutations of the ranking always have a lower bound quality of 0.

Results

We compare the performance of the crisp, state-of-the-art, diagnostic approach, presented in the Reasoning-based Diagnosis Section, with our fuzzy approach for the generated spectra. For more information regarding this approach, see [3].

In Figure 8, we compare the average Q_f for each test scenario. From the analysis of the plot we can see that the fuzzy approach always (on average) outperforms by the crisp approach. This is due to the fact that the fuzzy approach is able to successfully take advantage of the extra information to break the ties in the ranking (as shown in the example from Table 1) that occur when dealing with small numbers of erroneous transactions.

A more detailed analysis of the data (Figure 9) shows that our approach outperformed the crisp approach in 65% of the test cases. Moreover, in 94% of the cases our approach was at least as accurate as the classical approach. In the remaining 6% of the test cases the accuracy loss was due to (1) lack of observations, and (2) marginal

[7] By plausible we mean that all the candidate's components were at least activated once in an erroneous transaction.

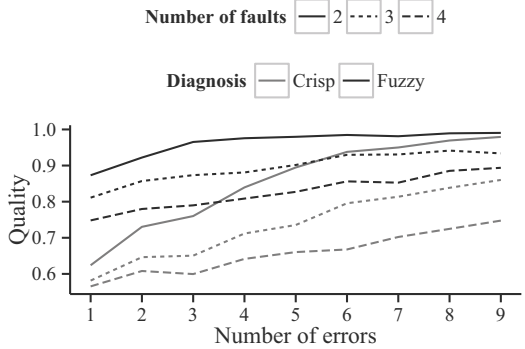


Figure 8: Benchmark Averages

variations in the posterior probability, still enough to make the relative ranking change. The overall average improvement of quality introduced by our algorithm was of $\Delta Q_f = 0.153$, representing a relative improvement of 21%. By performing a paired one-tailed T-test, we can ascertain that our approach introduced a relative improvement of 20%, with a 99% confidence interval.

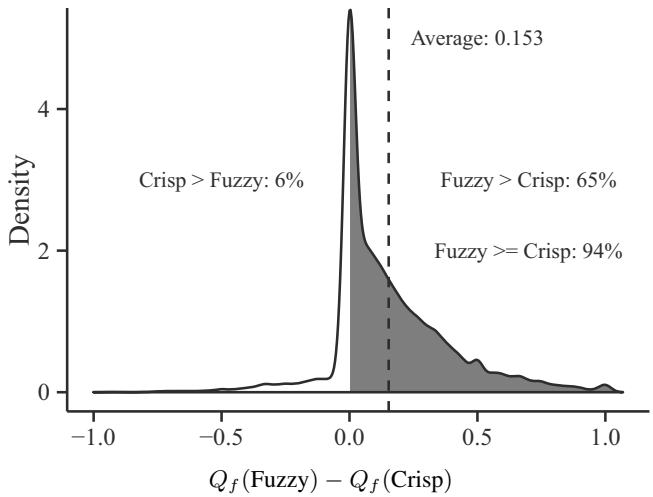


Figure 9: Quality improvement density plot

In Figure 10, we present a set of boxplots[8] comparing the quality distributions of both approaches for each test scenario. From the analysis of the plots we can see that not only the fuzzy approach has a better performance than the crisp approach, but also that the fuzzy approach distribution is much more skewed towards better quality results than the crisp approach. Additionally, we can see that the fuzzy approach exhibits a higher consistency (i.e., smaller inter-quartile range) than the crisp approach.

A final remark is that, with the increase of erroneous transactions, it appears that the crisp approach quality seems to converge towards to the same average quality as the fuzzy approach. This happens due to the fact that the information introduced by the occurrence of errors eventually compensates the limitations imposed by the crisp error abstraction.

[8] For each test scenario, the box corresponds to 2^{nd} and 3^{rd} quartiles (i.e., 50% of the cases), the vertical lines correspond to the 1^{st} and 4^{th} quartiles, and the small dashes correspond to test cases categorized as outliers. A test case is considered to be an outlier if its distance from the box is greater that $1.5 * IQR$ (inter-quartile range, i.e., the height of the box).

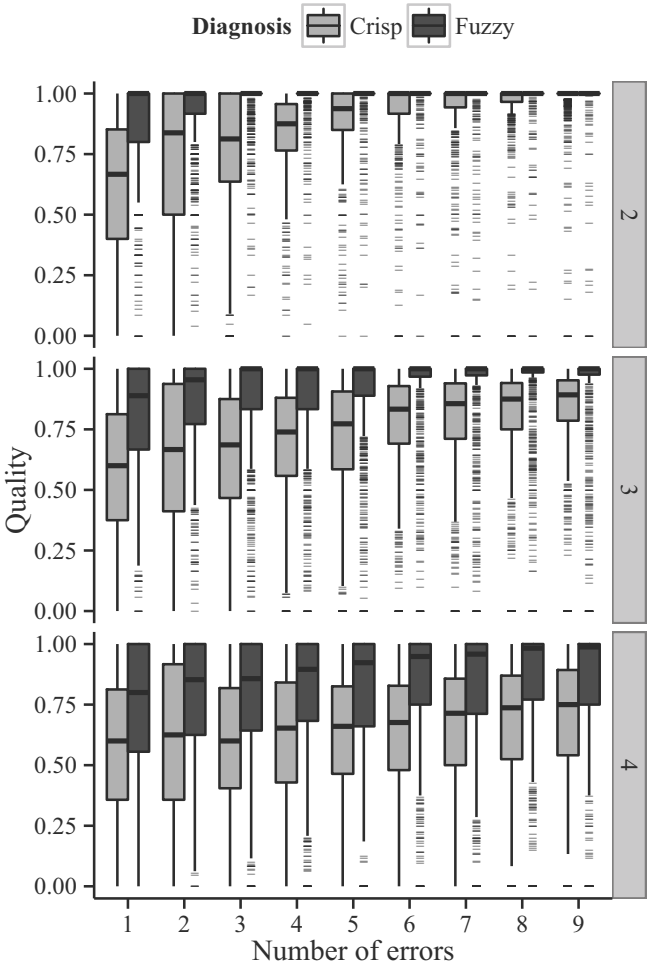


Figure 10: Benchmark Boxplots

Related Work

SFL and the improvement described in this paper occupy the ground between machine learning and model-based diagnosis [20]. Machine learning techniques [27] require a lot of training data which in the case of SFL is hit spectra. Hit spectra may look like a lot of data, however, the "per component" information is limited (we get n labels per component). Further, the per component pass/no pass information is Boolean and machine learning has to be trained for exponential number of multiple-faults, something that is not required in our approach. Last, if a lot of learning data is available, then machine learning easily overfits due to the Boolean nature of the features.

Model-based diagnosis is an attractive diagnostic approach because some frameworks are complete, others fast [11], and the complete ones use all available information to reason from symptoms to diagnoses. There are two fundamental problems with MBD approaches, however: (1) the exponential reasoning cost for complete algorithms and (2) the modeling problem for all MBD-based algorithms. Complex logical systems like computer programs cannot be mathematically modeled without encountering problems with decidability and halting.

Technique similar to ours, but applied to a different problem domain is the one of Kahuna [26]. There, the authors apply peer similarity to diagnose performance problems in map-reduce systems. In the black-box approach, the authors use learning from successfully passed tests. The Kahuna [26] black-box approach can be improved by using non-Boolean classification similar to the one proposed in this paper.

In [18], the authors apply the same concept to the diagnosis of failures in distributed file systems, such as PVFS or Lustre.

Computer programs nowadays fail mostly due to combination of faults. Single-faults are ruled-out with the help of classical debugging and unit-testing. Reasoning about multiple faults comes at steep computational price. Efficient lightweight fault localization techniques to diagnose software systems are PINPOINT [10], TARANTULA [17], and OCHIAI [2]. While extremely efficient, they do not reason in terms of multiple faults.

The approach in this paper improves state-of-the-art multiple-fault health estimation. This gives increased accuracy compared to simple similarity-based SFL which is optimal only in the case of single-faults. The improved ranking is shown and analyzed in [3], [19], and [6].

The algorithm described in this paper is very suitable for troubleshooting of large and complex, network systems. There are various latencies and degradations in these systems that make the pass/fail qualification of a test inadequate. Discrete event systems [24] are currently used for monitoring and diagnosis of networked systems but they have enormous computational cost and, unlike our approach, are more difficult to randomize and sample.

The approach in this paper is related to hybrid automata [15], an approach also originating in the control community. Hybrid automata merge Hidden-Markov Models and dynamic systems.

Conclusions

We presented a generalization to the classical reasoning-based approach that not only guarantees equal diagnostic quality when diagnosing functional failures but also improves the diagnostic quality when diagnosing performance degradation failures (soft failures).

The conducted synthetic benchmark showed that, for our setup with 27000 test cases, our approach improved the diagnostic quality in 65% of the cases and performed at least as good as the classical approach in 94% of the test cases. On average, the relative improvement introduced by our approach was of 20%, with a 99% confidence interval.

Future work includes the extension of existent error detection frameworks to include the fuzzy error abstraction proposed in this paper. Comparison with other approaches than the crisp approach (Barinel) has remained for future work, because our goal was to improve current crisp-based SFL approach (relevant due to its low-cost modeling and diagnostic effort, thus scaling to large, real software/hardware systems) in order to deal gracefully with non-functional errors. The existence of such a framework would enable a real-world validation of the proposed approach. Furthermore, the broader impact of this paper is that the approach paves the way to automatic oracles in the context of automatic test generation and self-healing systems.

Acknowledgements

We would like to thank Ion Matei, Lígia Massena, André Silva, and Alexandre Perez for the useful discussions about this work. This material is based upon work supported by the National Science Foundation under Grant No. CNS 1116848, and by the scholarship number SFRH/BD/79368/2011 from Fundação para a Ciência e Tecnologia (FCT).

REFERENCES

[1] Rui Abreu and Arjan J. C. Van Gemund. A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis. In *Proceedings of the* 8^{th} *Symposium on Abstraction, Reformulation, and Approximation*, SARA'09, 2009.

[2] Rui Abreu, Peter Zoeteweij, and Arjan J. C. Van Gemund. On the accuracy of spectrum-based fault localization. In *Proceedings of the* 2^{nd} *International Academic And Industrial Conference on Testing – Practice And Research Techniques*, TAICPART'07, pages 89–98, 2007.

[3] Rui Abreu, Peter Zoeteweij, and Arjan J. C. Van Gemund. A new bayesian approach to multiple intermittent fault diagnosis. In *Proceedings of the* 21^{st} *International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 653–658, 2009.

[4] Rui Abreu, Peter Zoeteweij, and Arjan J. C. Van Gemund. Spectrum-based multiple fault localization. In *Proceedings of the 2009 International Conference on Automated Software Engineering*, ASE'09, pages 88–99, 2009.

[5] Nuno Cardoso and Rui Abreu. MHS^2: A map-reduce heuristic-driven minimal hitting set search algorithm. In *Proceedings of the 2013 International Conference on Multicore Software Engineering, Performance, and Tools*, MUSEPAT'13, pages 25–36, 2013.

[6] Nuno Cardoso and Rui Abreu. A kernel density estimate-based approach to component goodness modeling. In *Proceedings of the* 27^{th} *AAAI Conference on Artificial Intelligence*, AAAI'13, 2013.

[7] John Carey, Neil Gross, Marcia Stepanek, and Otis Port. Software hell. In *Business Week*, pages 391–411, 1999.

[8] Paulo Casanova, David Garlan, Bradley Schmerl, and Rui Abreu. Diagnosing architectural run-time failures. In *Proceedings of the* 8^{th} *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS'13, pages 103–112, 2013.

[9] Cuiting Chen, Hans-Gerhard Gross, and Andy Zaidman. Improving service diagnosis through increased monitoring granularity. In *Proceedings of the* 7^{th} *International Conference on Software Security and Reliability*, SERE'13, pages 129–138, 2013.

[10] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, O Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, DSN'2002, pages 595–604, 2002.

[11] Alexander Feldman, Gregory Provan, and Arjan J. C. Van Gemund. Computing minimal diagnoses by greedy stochastic search. In *Proceedings of the* 23^{rd} *AAAI Conference on Artificial Intelligence*, AAAI'08, pages 911–918, 2008.

[12] Debanjan Ghosh, Raj Sharman, H. Raghav Rao, and Shambhu Upadhyaya. Self-healing systems - survey and synthesis. *Decision Support Systems in Emerging Economies*, 42(4):2164–2185, 2007.

[13] Alberto Gonzalez-Sanchez, Rui Abreu, Hans-Gerhard Gross, and Arjan JC van Gemund. Spectrum-based sequential diagnosis. In *AAAI'11*, 2011.

[14] Mary Jean Harrold, Gregg Rothermel, Rui Wu, and Liu Yi. An empirical investigation of program spectra. In *Proceedings of the 1998 Workshop on Program Analysis for Software Tools and Engineering*, PASTE'98, pages 83–90, 1998.

[15] Michael W Hofbaur and Brian C Williams. Mode estimation of probabilistic hybrid systems. In *Hybrid Systems: Computation and Control*, pages 253–266. Springer, 2002.

[16] Birgit Hofer and Franz Wotawa. Spectrum enhanced dynamic slicing for better fault localization. In *Proceedings of the* 20^{th} *European Conference on Artificial Intelligence*, ECAI'12, pages 420–425, 2012.

[17] James A. Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the* 20^{th} *International Conference on Automated Software Engineering*, ASE'05, pages 273–282, 2005.

[18] Michael P. Kasick, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Black-box problem diagnosis in parallel file systems. In *Proceedings of the* 8^{th} *Conference on File and Storage Technologies*, FAST, pages 43–56, 2010.

[19] Johan De Kleer. Diagnosing multiple persistent and intermittent faults. In *Proceedings of the* 21^{st} *International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 733–738, 2009.

[20] Johan De Kleer and Brian C Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[21] Johan De Kleer and Brian C. Williams. Readings in model-based diagnosis. In *Readings in Model-Based Diagnosis*, pages 100–117. Morgan Kaufmann Publishers Inc., 1992.

[22] W. Mayer and M. Stumptner. Model-based debugging using multiple abstract models. In *Proceedings of the 2003 International Workshop on Automated and Analysis-Driven Debugging*, AADEBUG'03, pages 55–70, 2003.

[23] E. Piel, A. Gonzalez-Sanchez, H-G. Gross, Arjan J. C. Van Gemund, and R. Abreu. Online spectrum-based fault localization for health monitoring and fault recovery of self-adaptive systems. In *Proceedings of the* 8^{th} *International Conference on Autonomic and Autonomous Systems*, ICAS'11, pages 64–73, 2012.

[24] Meera Sampath, Raja Sengupta, Stephane Lafortune, Kasim Sinnamohideen, and Demosthenis C Teneketzis. Failure diagnosis using discrete-event models. *Control Systems Technology, IEEE Transactions on*, 4(2):105–124, 1996.

[25] Friedrich Steimann, Marcus Frenkel, and Rui Abreu. Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ISSTA'2013, pages 314–324, 2013.

[26] Jiaqi Tan, Xinghao Pan, Eugene Marinelli, Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. Kahuna: Problem diagnosis for mapreduce-based cloud computing environments. In *Proceedings of the* 12^{th} *Network Operations and Management Symposium*, NOMS, pages 112–119, 2010.

[27] W Eric Wong and Yu Qi. Bp neural network-based effective fault localization. *International Journal of Software Engineering and Knowledge Engineering*, 19(04):573–597, 2009.

[28] Franz Wotawa. A variant of reiter's hitting-set algorithm. *Information Processing Letters*, 79(1):45–51, 2001.

[29] Lotfi Asker Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[30] Lotfi Asker Zadeh. Probability measures of fuzzy events. *Journal of Mathematical Analysis and Applications*, 23(2):421–427, 1968.

[31] Tom Zamir, Roni Stern, and Meir Kalech. Using model-based diagnosis to improve software testing. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1135–1141. AAAI Press, 2014.