

Multi-fault Diagnosis in Dynamic Systems

Natalia Odintsova, Irina Rish, Sheng Ma
IBM T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
{nodints,rish,shengma}@us.ibm.com

Abstract

In this paper, we consider the task of real-time event correlation and problem diagnosis. We present simple, linear-time approach that extends commonly used diagnostic techniques (such as codebook [2] and active probing [5]) to allow multiple fault diagnosis and change detection in dynamically changing systems (under certain assumptions about the fault occurrences). We demonstrate empirically the advantages of our approach.

1. Introduction

Fault diagnosis in dynamically changing systems is an important and challenging problem arising in systems performance management. Many commonly used approaches (e.g., *codebook* [2] and *active probing* [5]) simplify the problem by assuming a "static" system that does not change during the diagnosis cycle, and interpret the observed combination of events as a 'symptom' of some problem. However, in a truly dynamic systems, problems may occur and disappear while events are collected, leading to inconsistencies in observations (typically treated as "noise" [2]), and therefore increasing the diagnostic error.

Another challenge is the presence of multiple faults which become more likely with the growing size of a dynamic system. "Static" approaches (e.g. codebook) do not track the sequential occurrence of the faults (and repairs), and thus face the problem of diagnosing *simultaneous* multiple faults present in the system. However, general multifault diagnosis problem is known to be computationally hard (e.g., constrained-satisfaction formulation of [1] and probabilistic inference problem in Bayesian networks [4] are both NP-hard; also, handling multiple faults in a system of n components using codebook [2] or active probing [5] would require enumeration of up to 2^n fault combinations).

Herein, we propose a diagnostic approach that tracks system's changes over time and handles multiple faults incrementally; the resulting linear-time algorithm provides significant computational savings over general multifault approaches, while at the same improves the accuracy of "static" approaches. The price to pay is the restricting assumptions that failures and repairs happen one at a time and with a moderate frequency so that the diagnostic engine can process them sequentially. However, there appears to be a wide range of practical diagnostic problems satisfying these assumptions, where our approach provides a nice trade-off between accuracy and complexity of diagnosis.

We adopt a commonly used problem-determination framework known as *codebook* [2] (also called *dependency matrix* [5]). The codebook, or dependency matrix, $D = [d_{ij}]$ is a 0/1 matrix where columns correspond to possible problems, and rows correspond to observations (events, measurements). In the matrix, $d_{ij} = 1$ if problem p_j causes event e_i to happen, and $d_{ij} = 0$ otherwise. For example, in the context of diagnosis

by *probing* [5] columns correspond to particular components that can fail (e.g. nodes in a computer network, such as routers and servers, or software components such as web- and database applications), and rows correspond to end-to-end transactions, called *probes*, which involve ("go through") subsets of system components*. The problems are uniquely diagnosable if all columns are different.

In order to diagnose uniquely an arbitrary combination of multiple faults, the codebook approach would require an exponential number of columns. Another problem is that not all combinations of probe outcomes are even realizable, for example due to topological and other constraints. Thus, some multifault situations can be inherently unrecognizable because some components may become "shielded" by the failures of other components. Namely, we will define a component X as *shielded* by the failure of the component Y if all probes going through X go through Y as well.

We will now extend the codebook and active probing approaches in order to handle multiple and/or sequential faults.

2. Our Approaches

Generic Multifault. First, we consider a very simple algorithm for handling multiple faults (called *generic multifault*) that still assumes no change in the system state during the diagnosis cycle. Given the dependency matrix and the probes outcomes, the algorithm

1. Finds OK nodes: these are all the nodes through which at least one OK probe passed.
2. Finds failed nodes: these are the nodes through which any failed probe passed such that all other nodes on its path are OK nodes, as determined in step 1.
3. Finds *shielded* nodes: these are the nodes through which no probe goes other than those that go through any of already failed nodes. Thus, all those probes will return 'failure' and it is impossible to determine the state of such nodes, or, in other words, the nodes are 'shielded' by the failures of nodes found in step 2.
4. The remaining nodes are "possible failures", in the sense that certain combinations of their failures can produce the given set of probe outcomes.

Generic multifault that reports shielded nodes as failed does not miss any faults, although its false-positive error (the amount of OK nodes reported as faulty) can be high for certain dependency matrices. We will refer to this algorithm as "safe", oppose to "non-safe" that reports shielded nodes as OK. In fact, the generic multifault algorithm acknowledges the shielded nodes, and it is up to the user to decide how to interpret them. We will show in empirical section that, depending on the probability of fault in a system, we should prefer "safe" or "non-safe" version.

Sequential Multifault. We now extend the generic multifault approach to dynamic systems. The resulting algorithm, called *sequential multifault* is still linear in the number of probes, but has a lower diagnostic error because it keeps track of changes in the system: the inconsistencies in observations help to detect system change. The algorithm does not restrict the amount of faulty components in the system, but it assumes that only one

*In [5], an online probe selection approach was proposed that often requires much less number of probes (up to 70% in empirical studies). The approach, called *active probing*, selects probes on demand, choosing most-informative next probe depending on the outcomes of the previously observed probes.

change can happen at a time (i.e., failure or repair of one component), and that processing of each change is fast enough so that no other change occurs while the current change is being processed. At a very high-level, the algorithm performs the following monitoring loop:

```
initialize-system-state;
while (true)
    if current observation contradicts previous observations {
        diagnose change;
        report results;
    }
```

Particularly, the algorithm monitors changes in the system's states using two sets of probes: set for fix (i.e., repair) detection to monitor nodes that are known as failed, and set for failure detection to monitor nodes that are known to be OK. Just as for generic multi-fault, the algorithm has "safe" and "non-safe" ways of treating shielded nodes, which are compared in the empirical section. For details of the algorithm, see [3].

3. Empirical Results

We compared the two proposed algorithms versus single-fault active probing. The experiments were performed on randomly generated networks with 50 nodes and on a real-life router-level network. For synthetic networks, the probe sets were constructed by randomly choosing sources and then constructing shortest paths from each source to every other node. The number of probe stations varied from 1 to 10. The results were averaged over 30 trials for each number of sources. The simulations were run on sequences of 100 consecutive changes in the nodes' states, which were generated randomly, one change at a time (i.e., at each step, only one node can change its state). At each step, the change type - failure or repair - was chosen according to a "fault density" parameter that took values from 0 to 1 (fixed for each sequence, independent on the current system's state), which allowed to vary the average number of faults in the system over the whole sequence. We present results for the fault densities 10% and 50% (that is, when 10% and 50%, correspondingly, of all nodes were down on average).

We vary the number of probe stations, also called *sources*, and expect that increasing number of sources will decrease the amount of shielded nodes, since nodes may become reachable from multiple sources, and thus improve the accuracy of diagnosis. This indeed happens for multifault algorithms, while the single-fault one (active probing) is not affected – its error is always close to the fraction of faults in the system*.

The figures 1a and 1b compare the algorithms for two different regimes, i.e. low and high fault density, and suggest that (1) multifault algorithms are generally more accurate than the single-fault one, especially for higher fault densities, and (2) selection of "safe" versus "non-safe" version depends on the anticipated fault density. When the fault density

*Indeed, if there are N faults in the system, the single-fault algorithm will either find one of them, and miss all the others, or miss all of them, and instead diagnose some OK node as failure. In the first case, its false-negative error is $N-1$, with positive error being zero. In the second case, its false-negative error is N , and false-positive error is 1. So the lower bound on average total error is $N-1$, and the upper bound is $N+1$, where N is the number of actual faults in the system. This estimation holds for any implementation of the single-fault algorithm (active probing or codebook).

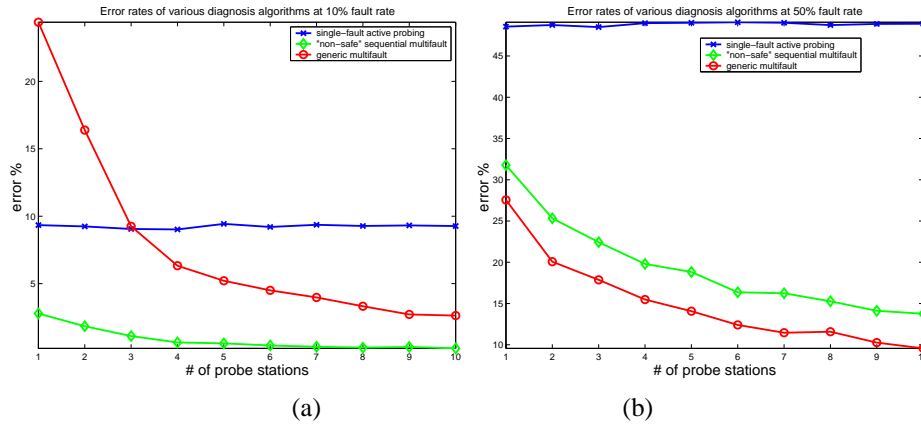


Figure 1: Error rates (a) at 10% fault density and (b) at 50% fault density.

is relatively low (e.g. 10%), the "non-safe" sequential algorithm yields the best results (Figure 1a) with the total error rate more than 3 times less than that of the single-fault algorithm, even for the case of a single source. The error rate of the "safe" multifault algorithms is relatively high when the number of sources is small, but quickly decreases with the number of sources, also outperforming the single-fault algorithm. When the fault density is high (50% in Figure 1b), any of the multifault algorithms performs better than the single-fault algorithm (Figure 1b), with the "safe" versions yielding the best results. So at such fault rates it is better to interpret the shielded nodes as failed, rather than OK*. For more detailed discussion and the results on real-life network (which looked quite similar to the simulated ones) see the extended version of this paper [3].

References

- [1] J. de Kleer and B.C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1), 1987.
- [2] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo. A coding approach to event correlation. In *Intelligent Network Management (IM)*, 1997.
- [3] N. Odintsova, I. Rish, and S. Ma. Multifault Diagnosis in Dynamic Systems. Technical Report RC23385, IBM T.J. Watson Research Center, 2004.
- [4] I. Rish, M. Brodie, and S. Ma. Accuracy vs. Efficiency Trade-offs in Probabilistic Diagnosis. In *Proceedings of AAAI-2002, Edmonton, Alberta, Canada, 2002*.
- [5] I. Rish, M. Brodie, N. Odintsova, S. Ma, and G. Grabarnik. Real-time Problem Determination in Distributed Systems using Active Probing. In *Proceedings of 2004 IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea, 2004*.

*It is interesting to note that, unlike low fault density, at high fault densities sequential "non-safe" multifault produces notably lower error than generic "non-safe" multifault algorithm. This difference can be explained by the fact that the sequential algorithm keeps track of the fault history, so if a node had failed and then later became shielded, sequential algorithm would "remember" this failure, rather than "blindly" declare all shielded nodes as OK, as the "non-safe" version of the generic multifault algorithm would do.