Real-time Problem Determination in Distributed Systems using Active Probing

Irina Rish, Mark Brodie, Natalia Odintsova,
Sheng Ma, Genady Grabarnik
IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
rish,mbrodie,nodints,shengma,genady@us.ibm.com

Abstract

We describe algorithms and an architecture for a real-time problem determination system that uses online selection of most-informative measurements – the approach called herein **active probing**. Probes are end-to-end test transactions which gather information about system components. Active probing allows probes to be selected and sent on-demand, in response to one's belief about the state of the system. At each step the most informative next probe is computed and sent. As probe results are received, belief about the system state is updated using probabilistic inference. This process continues until the problem is diagnosed. We demonstrate through both analysis and simulation that the active probing scheme greatly reduces both the number of probes and the time needed for localizing the problem when compared with non-active probing schemes.

Keywords

self-managing networks, real-time monitoring and problem determination, end-to-end response time measurements, AI techniques/probabilistic inference, information theory

1. Introduction

Accurate diagnosis in a complex, multi-component system by making inferences based on the results of various tests and measurements is a common practical problem. Developing cost-effective techniques for diagnosis in such systems requires that high accuracy be achieved with a small number of tests. In this work we present a generic approach to this problem and apply it specifically to the area of distributed systems management.

The key component of our approach is an "active" measurement approach, called *probing*. A probe is a test transaction whose outcome depends on some of the system's components; diagnosis is performed by appropriately selecting the probes and analyzing the results. In the context of distributed systems, a probe is a program that executes on a particular machine (called a probe station) by sending a command or transaction to a server or network element and measuring the response. The *ping* program is probably the most popular probing tool that can be used to detect network availability. Other probing tools, such as IBM's EPP technology ([7]), provide more sophisticated, application-level probes. For example, probes can be sent in the form of test e-mail messages, web-access requests, and so on.

Previous work on using probing for problem determination and diagnosis focused mainly on pre-planned, fixed probe sets, which are scheduled to run periodically [1, 7]. However, using pre-planned probe sets suffers from considerable limitations. Because the probe set is computed off-line, it needs to be able to diagnose all possible problems which might occur. However in practice many of these problems may in fact ever happen, and running the complete set of preplanned probes might be quite wasteful. Knowing which probes can be safely omitted can usually only be determined on-line by monitoring which problems in fact occur.

Another disadvantage of pre-planned probe sets is that because the probes run periodically at regularly scheduled intervals, there may be a considerable delay in obtaining information when a problem occurs. It is clearly desirable to detect the occurrence of a problem as quickly as possible. Furthermore, once the occurrence of a problem has been detected, additional information may be needed to diagnose the problem precisely. This information may not be obtainable from the results of the pre-planned probes - additional probes may need to be sent to obtain it. These probes should be appropriately selected "on-demand", based on the results of the previous probes.

Our works develops a methodology called **active probing** that addresses these limitations. This involves probing in an interactive mode, where probe results are analyzed to determine the most likely diagnosis, and then additional probes are selected and sent in order to gain further information. This process may repeat - once additional probe results are obtained, the diagnosis is refined, and, if necessary, more probes are selected, and so on, until the problem is completely determined. The idea of this approach is to "ask the right questions at the right time".

Active probing selects and sends probes as needed in response to problems that actually occur. It therefore avoids both the difficulty of constructing probes for all possible problems as well as the waste inherent in using probes for problems that in fact never occur. Furthermore, because probes are selected on-line to obtain further information about particular problems that have occurred, they need not circulate regularly throughout the entire network; instead they can be targeted quickly and directly to the points of interest. Thus fewer probes are needed than in a pre-planned approach, allowing for a considerable reduction in probing costs.

Clearly, active probing can be also combined with traditional approaches such as various event correlation techniques [11, 12, 9]. Our diagnostic engine accepts any input events, including probes, alarms and other messages, and performs appropriate inferences about the current system state. However, we add the ability of active measurement selection on top of such 'passive' inference capabilities.

In this paper, for illustration purposes, we will focus on the problem of fault diagnosis, where each component of the system is assumed to be either "OK" or "faulty" at any given time. However, our approach can be easily extended to a broader range of diagnostic problems, such as determining QoS levels. Indeed, in our applications, exceeding a threshold response time by a probe, or a particular application (component of a probe) was considered as a "soft" failure and treated similarly to "hard" failures (such as "server down") during the diagnostic process. Moreover, the approach can be also extended to handle multiple levels of a performance degradation by introducing several intermediate states for each component.

The outline of the paper is as follows. Section 2 provides the basic framework and notation. Section 3 summarizes our previous work on pre-planned probing [1], where the entire probe-set is computed before sending any probes. We show that this problem is NP-hard, and use an information-theoretic approach to develop linear and quadratic-time approximation algorithms. Section 4 presents an **active** probing approach, which selects probes based on the results of earlier probes. These ideas were presented very briefly in [3] - this paper extends that work by describing the system architecture and requirements needed for active probing (see Section 5), giving the details of our testbed implementation, and extending our previous experimental results for simulations and practical applications (Section 6). Our results show that active probing can greatly reduce the number of probes needed to diagnose problems. Related work is discussed in Section 7 and then we draw some Conclusions.

2. Framework and Notation

We have a set of nodes (components) $N = \{N_1, ..., N_n\}$, each of which can be either **"up"**, functioning correctly, or **"down"**, not functioning correctly. In a distributed system, the nodes may be physical entities such as routers, servers, and links, or logical entities such as software components, database tables, etc. The **state** of the system is denoted by a vector $\mathbf{X} = (X_1, ..., X_n)$ of Boolean variables, where X_i represents the state of node (component) N_i. Lower-case letters denote the values of the corresponding variables, e.g. $\mathbf{x} = (x_1, ..., x_n)$ denotes a particular assignment of node values. In general, there are 2^n different system states; however, in practice it can often be assumed that only k faults can occur simultaneously - indeed the case $k = 1$ is often sufficient.

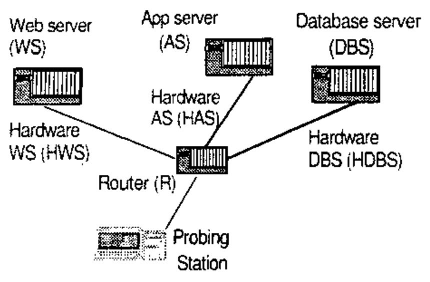
A *probe* is a method of obtaining information about the system components. The set of components tested by a probe p (i.e. the components p depends on) is denoted $N(p) \subseteq \{N_1, ... N_n\}$. A probe either succeeds or fails: if it succeeds, then every component it tests is up; it fails if **any** of the components it tests are down.

Fault localization attempts to determine the state of the system from the probe results. It is useful to introduce the notion of a **dependency matrix** to capture the relationships between system states and probes. Given any set of nodes $N = \{N_1, N_2, ..., N_n\}$ and probes $P = \{p_1, p_2, ..., p_r\}$, the dependency matrix $D_{P,N}$ is given by:

$$\begin{aligned} D_{P,N}(i,j) &= 1 \text{ if } N_j \in N(p_i) \neq \phi \\ &= 0 \text{ otherwise.} \end{aligned}$$

$D_{P,N}$ is an r-by-n matrix, where each row represents a probe and each column represents a node.

Figure 1a shows an example of a simple benchmark used in our proof-of-concept demo described in the subsequent sections. The benchmark contains a probing station, which sends various probes through a common router to a Web Server (WS), Application Server (AS), and Database Server (DBS). Note that WS, AS and DBS represent the OK/not OK state of the corresponding applications running on these machines, while HWS, HAS and $HDBS$ denote "hardware" problems with WS, AS, and DBS, respectively (in our case, HWS is OK if WS can be reached by *ping* command; however, the web server application may not be running, and thus WS is not OK). Also, R will denote the state of



(a)

Dependency matrix

Probe \ Problem	WS	AS	DBS	R	HWS	HAS	HDBS	NF
pWS	1	1	1	1	1	1	1	0
pAS	0	1	1	1	0	1	1	0
pDBS	0	0	1	1	0	0	1	0
pingR	0	0	0	1	0	0	0	0
pingWS	0	0	0	1	1	0	0	0
pingAS	0	0	0	1	0	1	0	0
pingDBS	0	0	0	1	0	0	1	0

Probes:

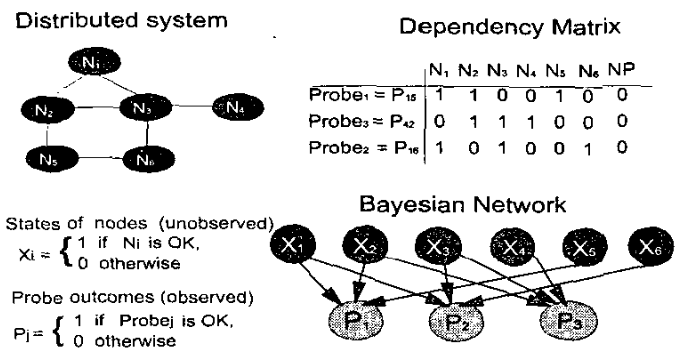
pWS - Web Page access, **pAS**- Application Server access, **pDBS** - Database query, **pingR** – ping router, **pingWS** – ping Web Server, **pingAS** - ping Application server, **pingDBS** - ping Database Server

(b)

Figure 1: (a) A simple benchmark distributed system with one probe station and 7 probes; (b) dependency matrix for the system in (a).

the router, while NF corresponds to no-failure situation. The dependency matrix shown in Figure 1b includes the following probes:
"main" probe called pWS, attempts to open a web page on WS, which also runs an application on AS, which in its turn sends a query to a database on DBS. The outcome of this probe depends on the state (i.e., OK/not OK) of all components, i.e. WS, HWS, AS, HAS, DBS, and $HDBS$, as well as on the state of the router R. Thus, the row of the probe pWS contains ones in all columns (i.e., fails if any of these components fail).
- probe pAS calls an application on AS which sends a query to the database on DBS; thus the probe depends on the states of AS, HAS, DBS, $HDBS$, R.
- probe $pDBS$ sends a query to the database on DBS, and thus depends on DBS, $HDBS$ and R.
- probes $pingR$, $pingWS$, $pingAS$ and $pingDBS$ are simply $ping$ commands to the router and the corresponding servers.

In this example we are interested in single node failures and the case of no failure anywhere in the network. In general, multiple simultaneous failures can be handled by adding columns to the dependency matrix, each additional column representing a simultaneous failure of multiple nodes. Unfortunately, this approach requires at least the number of probes that is exponential in the number of possible faults; an alternative approach is to introduce, if possible, more probes that test each node directly. Optimal probe selection for multi-fault diagnosis is very similar to the *group testing* problem [6], extensively studied in the literature. The objective of group testing is to find all 'active' items (e.g., sick patients) in a given set of items by using a sequence of disjunctive tests (where the test outcome is the disjunction of the inputs). However, the presence of constraints such as



Distributed system
N1
N2
N3
N4
N5
N6
Dependency Matrix
Bayesian Network
States of nodes (unobserved)
Xi = { 1 if Ni is OK, 0 otherwise
Probe outcomes (observed)
Pj = { 1 if Probej is OK, 0 otherwise
X1
X2
X3
X4
X5
X6
P1
P2
P3

	N_1	N_2	N_3	N_4	N_5	N_6	NP
$Probe_1 = P_{15}$	1	1	0	0	1	0	0
$Probe_3 = P_{42}$	0	1	1	1	0	0	0
$Probe_2 = P_{16}$	1	0	1	0	0	1	0

Figure 2: A mapping from dependency matrix to a Bayesian network.

network topology makes probe selection a more complicated problem than the unconstrained group testing. We leave the treatment of optimal multi-fault probe selection out of scope of this paper.

While the dependency matrix is a common representation, we can also use a probabilistic model which describes a joint distribution over the system states. For example, the dependency matrix can be easily mapped to a two-layer Bayesian network [14] where the components X_i correspond to upper-level variables, the probes P_j correspond to the lower-layer variables, and subsets $N(P_j)$ are the set of *parents* of variable P_j in the Bayesian network (see Figure 2). Then the joint probability distribution can be written as

$$Pr(\mathbf{x}, \mathbf{p}) = \prod_{i=1}^{n} Pr(x_i) \prod_{j=1}^{m} Pr(p_j | N(p_j)), \tag{1}$$

assuming that the state variables X_i's are marginally independent, and that each probe outcome depends only on the components tested by this probe. $Pr(x_i)$ specifies the prior probabilities of the system states, while the *conditional probabilities distributions (CPDs)* $Pr(p_j|N(p_j))$ describe the dependency of probe outcomes on the components tested.

In the absence of noise in the probe outcomes, all CPDs are deterministic functions. Since a probe succeeds if and only if all its components are OK, a probe outcome is a logical-AND function of its components, i.e. $P_i = X_{i_1} \wedge ... \wedge X_{i_k}$, where $\wedge$ denotes logical AND, and $X_{i_1}, ..., X_{i_k}$ are all the nodes probe P_i goes through. In practice, however, this relationship may be disturbed by noise in the measurements. For example, a probe can fail even though all the nodes it goes through are OK (e.g., due to packet loss). Conversely, there is a chance that a probe succeeds even if a node on its path has failed (e.g., dynamic routing may result in the probe following a different path). Such uncertainties yield a so-called *noisy-AND* model; see [15] for more detail on noisy-AND Bayesian network model for problem diagnosis.

Bayesian network can be used for *probabilistic inference*[14], such as *belief updating*, i.e. finding $Pr(\mathbf{Z}|\mathbf{Y} = \mathbf{y})$, the posterior probability of set of variables $\mathbf{Z}$ given observations of some other variables $Pr(\mathbf{Y} = \mathbf{y})$. For example, we may want to update the fault probability of every single node given the probe outcomes, or we may want to find a single most-likely state of the systems (combination of faults). We will use belief updating as a part of active-probing-based diagnosis. See [15] for more details on problem diagnosis in distributed systems using Bayesian networks.

3. Pre-Planned Probing

We now give an overview of pre-planned probing algorithms proposed earlier in [1]. We reformulate them in a unified information-theoretic framework that also naturally extends to the online, active version.

In pre-planned probing, we want to compute the smallest probe-set such that each system state will produce a different set of probe outcomes, allowing the state to be uniquely determined. Since columns can be added to represent the case of no failure or multiple simultaneous failures, this can be formulated as finding the smallest probe set such that every column of the dependency matrix is unique, since in that case exactly which state has occurred can be determined from the outcomes of all the probes.

Probe-set selection: Find P^* which minimizes $|P'|$, where $P' \subseteq P$ is such that every column in $D_{P',N}$ is unique.

Proposition 1 *Probe-set selection is NP-hard.*

Proof: Probe-set selection can be shown to be NP-hard via a reduction from 3-Dimensional Matching (defined in [8]). We omit the details due to space considerations; see [2] for details.

We now present two approximation algorithms for probe-set selection - greedy search and subtractive search. "Greedy" search starts with the empty set and adds at each step the "best" of the remaining probes. The "best" probe is the one which maximizes the information gained about the system state, in the sense defined precisely below. "Subtractive" search starts with the complete set of available probes, considers each one in turn, and discards it if it is not needed. Neither algorithm is optimal in general - experimental results comparing their performance with the true minimum probe set size are given in Section 6.

The *greedy search* approach chooses the next probe by maximizing the information gained about the system state $\mathbf{X}$, given the previous probes. Formally, we are looking for a probe

$$Y^* = \arg\max_{Y \in \mathbf{P} \setminus \mathbf{P}'} I(\mathbf{X}; Y|\mathbf{P}'), \tag{2}$$

where $I(\mathbf{X}; Y|\mathbf{P}')$ is the conditional mutual information of $\mathbf{X}$ and probe Y, given the previously selected probes $\mathbf{P}'$. Since $I(\mathbf{X}; Y|\mathbf{P}') = H(\mathbf{X}|\mathbf{P}') - H(\mathbf{X}|Y, \mathbf{P}')$, where $H(\mathbf{X}|Y)$ is the conditional entropy of $\mathbf{X}$ given Y (see [4]), the most-informative probe Y^* minimizes the conditional entropy of $\mathbf{X}$, i.e. the amount of uncertainty about the system state.

The algorithm is shown in Figure 3a. If the initial set of available probes P is of size r, $O(r^2)$ conditional entropy calculations are required, since at each step the information

```
Probe-Set Selection: Greedy Search

Input: A set of available probes $\mathbf{P}$ and a prior
distribution over system states $Pr(\mathbf{X})$.
Output: A subset $\mathbf{P}' \subseteq \mathbf{P}$ of probes.
Initialize: $\mathbf{P}' = \emptyset$
do 1. select most-informative next probe:
      $Y^* = \arg\max_{Y \in \mathbf{P} \setminus \mathbf{P}'} I(\mathbf{X}; Y|\mathbf{P}')$
   2. update probe set: $\mathbf{P}' = \mathbf{P}' \cup \{Y\}$
while $\exists Y \in \mathbf{P} \setminus \mathbf{P}'$ such that $I(\mathbf{X}; Y|\mathbf{P}') > 0$
Return $P'$.
```

```
Probe-Set Selection: Subtractive Search

Input: A set of available probes $\mathbf{P}$ and a prior
distribution over system states $Pr(\mathbf{X})$.
Output: A subset $\mathbf{P}' \subseteq \mathbf{P}$ of probes.
Initialize: $\mathbf{P}' = \mathbf{P} = \{p_1, p_2, \ldots, p_r\}$.
for $i = 1$ to $r$
    remove probe if it is not needed, i.e.
    if $I(\mathbf{X}; p_i|\mathbf{P}' \setminus \{p_i\}) = 0$.
    then $\mathbf{P}' = \mathbf{P}' \setminus \{p_i\}$
Return $\mathbf{P}'$.
```

Figure 3: (a) Greedy Search for Probe-Set Selection. (b) Subtractive Search for Probe-Set Selection.

gain obtained by each of the remaining probes must be computed. Note that

$$H(\mathbf{X}|Y, \mathbf{P}) = -\sum_{\mathbf{x}} \sum_{y_j} \sum_{\mathbf{p}} Pr(\mathbf{x}, y, \mathbf{p}) \log Pr(\mathbf{x}, y, \mathbf{p}) \tag{3}$$

so computing the information gain can be quite costly in the general case, as it requires summation over all non-zero-probability states and outcomes of the current probe set and the next probe.

However in the case of only one component failing at a time and with equal prior probabilities of failure (including the case of no failure at all), the computation can be considerably simplified. A probe set cannot distinguish failures in nodes whose columns in the dependency matrix are identical. Since this is an equivalence relation between nodes, it induces a decomposition of the nodes into an exhaustive collection of disjoint subsets, and it is easy to show that:

$$H(\mathbf{X}|Y, \mathbf{P}) = \sum_{i=1}^{k} \frac{n_i}{n} \log n_i \tag{4}$$

where n is the total number of nodes and n_i is the number of nodes in the i'th subset of the decomposition induced by P.

This expression has a natural interpretation. Since there are n_i states in the i'th subset and each probe has two possible outcomes, at least $\log n_i$ additional probes are needed to further decompose the i'th subset into singletons, thereby enabling any single node failure to be diagnosed. Since the true failure lies in the i'th subset with probability n_i/n, the conditional entropy is simply the expected minimal number of additional probes needed to localize the failure.

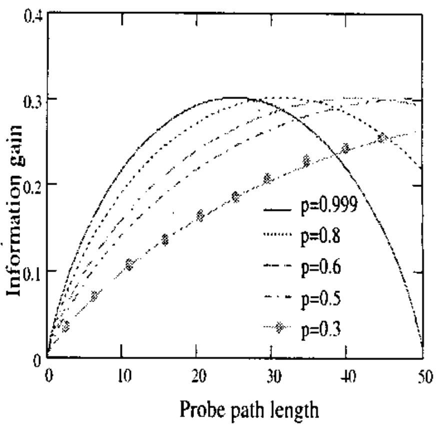
The greedy algorithm can of course be generalized by adding the best subset of t of the remaining probes at each step, requiring $O(r^{t+1})$ conditional entropy calculations.

The *subtractive search* algorithm starts with the complete set of available probes, considers each one in turn, and discards it if it is not needed, i.e. if removing it does not result in any loss of information about the system state. The algorithm is shown in Figure

```
Active Probing

Input: A set of available probes $\mathbf{P}$ and a prior distribution over system states $Pr(\mathbf{X})$.
Output: A set $\mathbf{P}_a$ of probes and their outcomes, posterior distribution $Belief(\mathbf{X})$, its support $\mathbf{S}$.
Initialize: $Belief(\mathbf{X}) = Pr(\mathbf{X})$, $\mathbf{P}_a = \emptyset$,
$\mathbf{S}$ = support of $Pr(\mathbf{X})$.
do
    1. select current most-informative probe:
       $Y = \arg\max_{Y \in \mathbf{P} \setminus \mathbf{P_a}} I(\mathbf{X}; Y|\mathbf{P}_a)$
    2. execute $Y$; it returns $Y = y$ (0 or 1)
    3. update $\mathbf{P}_a = \mathbf{P}_a \cup \{Y = y\}$
    4. update $Belief(\mathbf{X}) = Pr(\mathbf{X}|\mathbf{P}_a)$
while $\exists Y \in P$ such that $I(\mathbf{X}; Y|\mathbf{P}_a) > 0$
Return $\mathbf{P}_a$, $Belief(\mathbf{X})$,
$S$ =support of $Belief(\mathbf{X})$.
```

(a)



(b)

Figure 4: (a) Active Probing algorithm for probabilistic diagnosis with most-informative probe selection; (b) RAIL system architecture. (b) Information gain as a function of probe's 'effective' length (the number of nodes on the probe's path having non-zero fault probability) for various probabilities p of a single fault occurring in a system that contains 50 components.

3b. Each probe is considered only once, so $O(r)$ conditional entropy computations are required.

4. Active Probing

Here we extend the probing paradigm to allow for **active probing**, where the selection of later probes depends on the results of earlier probes. Probe-stations issue probes which traverse different parts of the network. The results of the probes are analyzed to infer what problems might be occurring. If additional information is needed in order to locate the problem, the most useful probes to send next are determined and sent. When additional probe results are received further inferences are made and the process repeats until the fault is localized.

The advantage of this approach is that fewer probes can be used than if the entire probe set has to be pre-planned. However additional inferential machinery is required. We describe an algorithm for active probing and then present experimental results that show that active probing can greatly reduce the number of probes needed to perform fault localization.

An active probing algorithm is described in Figure 4a. It takes as input a set of probes P available for selection, and a prior distribution $Pr(\mathbf{X})$ over system states. The approach is very similar to the greedy probe-set selection algorithm presented above, except that each selected probe is sent, the result obtained, and the probability distribution over the system state is updated. The algorithm maintains the current *belief* about the system state,

$Belief(\mathbf{X}) = Pr(\mathbf{X}|\mathbf{P_a})$ where $\mathbf{P_a}$ is the current set of probes and their outcomes. The prior distribution is used to initialize $Belief(\mathbf{X})$.

The active-probing diagnosis algorithm works as follows. It selects the next probe to run (step 1) and waits for the results (step 2), then it updates both the set of active probes executed (step 3) and the current belief about the system's state (step 4). Steps 1-4 are repeated until no more information can be obtained about the system state, and thus the diagnosis cannot be improved. The algorithm outputs a set of active probes P_a that were actually used during diagnosis (which often turns out to be significantly smaller than the original set P), the posterior distribution over components after receiving the probe outcomes, $Pr(\mathbf{X}|P_a)$, and the *support* of the distribution (the components with non-zero-probability).

In case of single fault in a system, the support contains no more than n nodes and thus $Pr(\mathbf{X})$ can be represented in $O(n)$ space. Once again simple expressions can be obtained for certain priors - for example a $1-p$ probability of no-fault, and uniformly distributed probability mass p among single component failures, $Pr(X_i = 0, X_j = 1 \forall j \neq i) = p/n$. Figure 4b plots the information gain of a probe as a function of its *effective length* k (*effective* length of a probe is the number of nodes on probe's path that currently have non-zero fault probability), for $n = 50$ nodes and for various fault priors. We see that it is more beneficial to send probes with larger effective length if the probability of fault p is small. However, once a fault is detected ($p = 1$), the most informative probe (i.e. a probe attaining the maximum information gain) is one whose effective length is closest to half the number of nodes that are possibly faulty.

The active probing algorithm can be also applied to a generic multiple-fault case; however, its complexity increases with an increasing number of simultaneous faults and depends on the efficiency of representing the joint probability $Pr(\mathbf{X})$ and the efficiency of probabilistic inference and information-gain computation required for active probe selection.

5. Real-time Diagnosis and Probe Selection Systems

The algorithms described above were used as a basis for building a system a system for probe set selection and dependency matrix analysis, called *DMA (Dependency Matrix Analysis tool)*, and a system for real-time diagnosis called *RAIL (Real-Time Active Inference and Learning)*.

DMA systems functions as follows. Given as an input a dependency matrix, the analyzer will first check for obviously redundant probes, e.g. delete identical copies of a probe (we encountered such cases in practice). DMA will also find all subsets of indistinguishable nodes and group them accordingly into meta-nodes. Then, DMA applies the following analysis to the resulting matrix:
- if exact search is not too expensive, DMA finds all optimal probe subsets using exact pre-planned probing;
- it finds a suboptimal probe subset using greedy search;
- it finds a probe set for problem detection only, using a greedy approach such as in active probing, until there are no 'uncovered' nodes left;

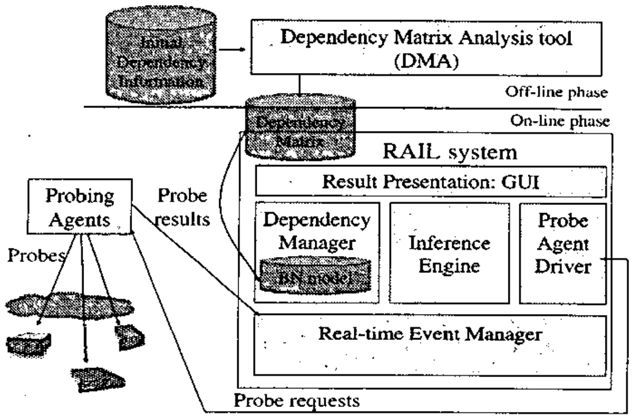


Figure 5: RAIL system architecture.

- DMA analyzes the effectiveness of active probing on the given dependency matrix by simulating a single fault, as well as no-fault situation, and computing the number of active probes required; it also computes an average number of active probes.

Next, we implemented the active probing approach within the prototype real-time diagnostic system called RAIL. The system architecture is shown in Figure 5. The real-time diagnosis engine obtains the input through the Real-Time Event Manager (REM) which is a generic component able to process not only incoming probes but various other event types: thus the diagnostic engine is not probe-specific. In our particular application which uses the IBM's End-to-End Probing Platform (EPP), the input probe outcomes are obtained from the EPP probe stations, processed by REM, and submitted to the diagnostic engine which updates its beliefs about the system states, and requests an active probe, if needed, using the Probe Agent Driver component, which 'talks' to EPP. The dependency matrix information is maintained and updated by Dependency Manager (DM), which obtains the initial matrix from the DMA tool; in the future, the dependency matrix will be constantly updated by the learning component(thus making RAIL a truly Real-time active inference and *learning* tool) if dependencies change dynamically, e.g., due to dynamic routing, addition/deletion of nodes, and other reasons.

Real-time diagnosis engine can be easily extended to handle multiple faults and repairs sequentially, assuming a single failure at a time. Once a failed probe is obtained which is not explained by any of the currently diagnosed failures, an active probing diagnosis is called. It identifies a single fault (if the remaining probe set permits unique diagnosis), and the corresponding node is deleted from the dependency matrix together with all probes going through the node. The system waits for the next probe, and so on. We assume that in case of repair of a node, an immediate message is sent to RAIL (this assumption greatly simplifies the sequential diagnosis algorithm). The beliefs about the system state are then updated, and sequential diagnosis continues. Note that in sequential multi-fault diagnosis,

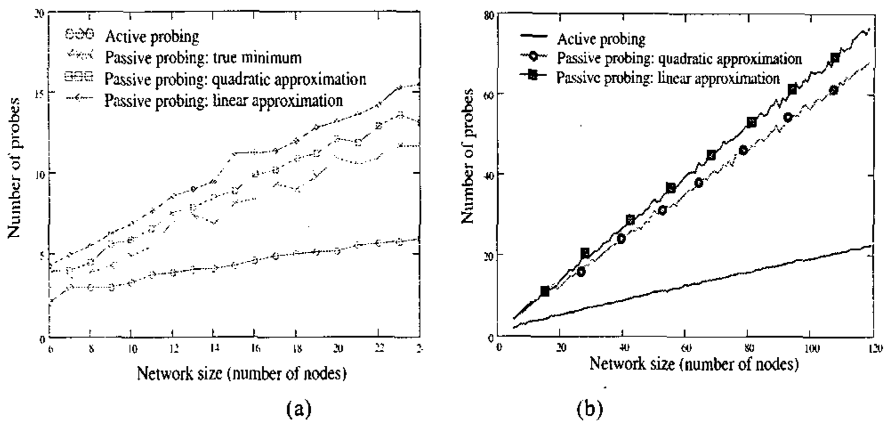


Figure 6: Active versus pre-planned probing results for randomly generated networks: simulation results on (a) small-scale and (b) large-scale networks.

some of the faulty nodes can make the other nodes unreachable and thus impossible to diagnose (e.g., in Figure 1, HAS failure makes AS unreachable). This indicates the necessity of developing more informative probe sets for handling multiple faults – yet another direction for future work.

6. Empirical Results

This section examines the empirical behavior of both pre-planned and active probing. For pre-planned probing the approximation algorithms find a probe set which is very close to the true minimum set size, and can be effectively used on large networks where finding the true minimum by exhaustive search is impractical. Active probing greatly reduces the number of probes needed, although at the expense of a more complex interactive inferencing system, as described above.

6.1 Simulated Networks

For each network size n, we generated twenty random networks with n nodes by randomly connecting each node to four other nodes. The probe stations are selected randomly. The probes follow the least-cost path from each probe station to each node.

The states to diagnose are any single node being down or no failure anywhere in the network. Each node has the same prior probability of failure, and there is no noise in the probe results. Note that in this case n probes are sufficient, because one can always use just one probe-station and probe every single node.

Exhaustive search is performed to find the true minimum size probe set. Then linear-time subtractive search and quadratic-time greedy search are used to find probe sets. Active probing algorithm is evaluated as follows. For each network, we simulate all possible fault scenarios (i.e., a fault at each node, and the no-failure situation), and compute an

	# of nodes	# of probes	Pre-planned probes (exact)	Pre-planned probes (greedy)	Pre-planned probes (detect)	Active Probing: min	Active Probing: max	Active Probing: average	Savings % active vs exact
O1	19	22	15	15	8	3	8	4.9	67%
O2	38	32	27	27	17	3	17	7.7	72%
O3	34	29	24	24	16	3	16	7.5	69%
G1	8	8	5	5	3	3	4	3.2	36%

Figure 7: Active probing results on several practical problems.

average number of active probes needed for diagnosis in this network. Finally, for every probing method, we average the results over all networks of given size and report them in Figure 6.

The results in Figure 6a (small-scale networks) indicate that the approximation algorithms for finding the smallest probe set perform well and are much closer to the true minimum set size than to the upper bound of n probes and also demonstrate the considerable improvement resulting from active probing when compared with pre-planned, or "passive", probing. In Figure 6b the approximation and active probing algorithms are extended to larger networks for which finding the true minimum is impractical. The active probing demonstrates more than 60% improvement over the pre-planned probing.

6.2 Practical Applications

In Figure 7, we report the results on several real probing applications. The problem $G1$ is a relatively small testbed for probe analysis, while the set of problems $O1$, $O2$ and $O3$ relates to several networks supporting e-business applications, which include many servers and routers, and its performance and availability depends on a large number of software components (such as various databases, etc). For the purposes of high-level, overall network diagnosis, only a set of aggregate components is specified (e.g., a particular network 'cloud', or a firewall of a specific company are considered as diagnosable components). A set of probes was manually selected by an expert for the case of single fault localization. The table shows the number of nodes and the number of probes in the original network after our initial processing of dependency matrix that eliminated repetitive probes and merges indistinguishable nodes. The next two columns show the minimum number of probes found in a pre-planning phase by exhaustive and by greedy search, respectively. Then the next column shows the minimum number of probes (found in greedy way) necessary for fault detection only (i.e., simply the probes 'covering' all nodes). Finally, we show the minimum and the maximum number of probes required by active probing to diagnose a single fault, and the average such number over all possible faults.

Our approach was quite successful for these applications. For example, in problem $O3$ having 34 nodes and 29 probes, the probe-set selection algorithm found that the minimum number of probes required for single fault localization is only 24 probes, a saving of

17%. Approximation algorithms were optimal or nearly optimal: greedy search returned 24 probes, while subtractive search found 25 probes (we only show the results of greedy search in the table since it was always superior to subtractive search). Finally, the most impressive results were obtained by active probing. The number of probes needed never exceeded 16 probes; on average, active probing required only 7.5 probes, versus 24 probes used by pre-planned probing, which yields savings of 69% (and of almost 74% if the initial probe set is considered). Similarly, active probing versus pre-planned one (and active probing versus initial probe set size) saved on average 67% (77%) probes on problem $O1$, 72% (76%) probes on problem $O2$, and 36% (%60) probes on problem $G1$.

7. Related Work

Previous work [1, 13] studied the probe selection problem for the purpose of network management. This paper develops a more general framework for problem determination using probes, proves the NP-hardness of the probe-set selection problem, develops the active probing approach and demonstrates its advantages in reducing probe-set size.

Event correlation [12, 9, 11] for identifying root-causes has long been recognized as a critical issue in the system management domain. Problem determination is performed by analyzing alarms emitted by devices when a significant situation occurs. Unlike the probing scheme, alarms are "reactive" to a situation and this requires intensive instrumentation, only possible in a tightly managed environment. The probing approach uses test transactions that can be built easily without touching the existing devices.

Nonetheless, event correlation has many similarities to our work. The formulation of problem diagnosis as a "decoding" problem, where "problem events" are decoded from "symptom events", was first proposed by [11]. Our approach uses an active probing approach versus a "passive" analysis of symptom events; namely [11] selects codebooks (a combination of symptoms encoding particular problems) from a specified set of symptoms, while we actively construct those symptoms (probes), a much more flexible approach. Another important difference is that [11] lacks a detailed discussion of efficient algorithms for constructing optimal codebooks.

Bayesian network approaches to fault diagnosis in computer networks and distributed systems were also considered previously (e.g., [10, 15]). However the active test selection approach as formulated here was not addressed in that work. Most informative test selection for multi-fault diagnosis was considered in AI literature related to other applications [5], however, to the best of our knowledge, combining probabilistic inference with online selection of most-informative probes appears to be a novel approach in the area of distributed systems diagnosis via end-to-end probing.

8. Conclusion

In this paper, we address the problem of real-time diagnosis in distributed systems using test transactions, or probes. Our main objective is to speed up real-time diagnosis by minimizing set of measurements, such as probes, while maintaining high diagnostic accuracy.

We consider two probing approaches: pre-planned and active. Finding the smallest pre-planned probe set is NP-hard, but approximation algorithms perform well. Active probing considerably reduces the number of probes needed but requires a more complicated technology to determine which probes to send. Our work suggests the merits of a combined approach, which uses pre-planned probes to detect when a problem occurs and then invokes active probing to locate the problem precisely.

Directions for future work include real-time diagnosis with changing network state and intermittent faults, handling dynamic routing and lack of precise knowledge of the probe path, and adapting to non-stationary behavior of the system using on-line learning that yields a dynamic, adaptive, probing strategy. We feel that adaptive, cost-effective techniques for problem determination will become increasingly important if new-generation IT systems are to be capable of self-management and self-repair.

References

[1] M. Brodie, I. Rish, and S. Ma. Optimizing probe selection for fault localization. In *Distributed Systems Operation and Management*, 2001.

[2] M. Brodie, I. Rish, S. Ma, A. Beygelzimer, and N. Odintsova. Strategies for Problem Determination Using Probing. Technical report, IBM T.J. Watson Research Center, 2002.

[3] M. Brodie, I. Rish, S. Ma, and N. Odintsova. Active probing strategies for problem diagnosis in distributed systems. In *Proceedings of the The Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico*, 2003.

[4] T.M. Cover and J.A. Thomas. *Elements of information theory*. New York:John Wiley & Sons, 1991.

[5] J. de Kleer and B.C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1), 1987.

[6] D-Z. Du and F.K. Hwang. *Combinatorial Group Testing and Its Applications (2nd edition)*. World Scientific, 2000.

[7] A. Frenkiel and H. Lee. EPP: A Framework for Measuring the End-to-End Performance of Distributed Applications. In *Proceedings of Performance Engineering 'Best Practices' Conference, IBM Academy of Technology*, 1999.

[8] M.R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-completeness*. W.H. Freeman and Co., San Francisco, 1979.

[9] B. Gruschke. Integrated Event Management: Event Correlation Using Dependency Graphs. In *Distributed Systems Operations and Management*, 1998.

[10] JF. Huard and A.A. Lazar. Fault isolation based on decision-theoretic troubleshooting. Technical Report 442-96-08, Center for Telecommunications Research, Columbia University, New York, NY, 1996.

[11] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo. A coding approach to event correlation. In *Intelligent Network Management (IM)*, 1997.

[12] A. Leinwand and K. Fang-Conroy. *Network Management: A Practical Perspective, 2nd Edition*. Addison-Wesley, 1995.

[13] H.C. Ozmutlu, N. Gautam, and R. Barton. Zone recovery methodology for probe-subset selection in end-to-end network monitoring. In *Network Operations and Management Symposium*, pages 451–464, 2002.

[14] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[15] I. Rish, M. Brodie, and S. Ma. Accuracy vs. Efficiency Trade-offs in Probabilistic Diagnosis. In *Proceedings of the The Eighteenth National Conference on Artificial Intelligence (AAAI-2002), Edmonton, Alberta, Canada*, 2002.