INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 10, NO. 2, JUNE 2005, 107-113

Proactive Probing and Probing On Demand in Service Fault Localization

Zhixiong CHEN

Abstract- **This paper proposes Proactive Probing and Probing on Demand (pPOD) approach to monitor the health of a serviceable component system (SCS) proactively and to localize faulty serviceable component on demand. In general, many high-level web services have dependency on other web services and on the basic network infrastructure. The health of such high-level services relies on the health of all these services and the network infrastructure. Therefore, we can select a small set of such high level web services and monitor them by probing their health periodically. This application-level performance monitoring mechanism can greatly reduce the overhead of satisfying service-level objectives. When abnormal events occur, we are able to follow the dependency path to localize possible failures based on the outcomes of incremental on demand probing. The proposed approach effectively reduces the resource and traffic cost in performance monitoring and speeds up the diagnosis process when compared with traditional performance monitoring and fault localization methods.**

***Index Terms*—Performance Monitoring; Fault Localization; Proactive Probing; Reactive Diagnosis; Spatial and Temporal locality.**

1. INTRODUCTION

User-perceived web service performance is an important part of performance management in a web service environment. These services are usually end-to-end user applications and depend on other services and on network infrastructures in which services are hosted. For example, a simple online inventory inquiry request depends on many hardware and software components such as user-side PC machine, a browser, Domain Name Service (DNS), routers and hubs, server side machines, web servers, application servers and database servers. Any abnormal performance of these components will cause on the user side a longer response time or even failure. By monitoring the quality and performance of these web services, we can actually monitor all the elements upon which these services depend.

When an abnormal or failure event happens, we, however, would like to diagnose each component along the path in order to localize a faulty component or components.

This technology is generally termed as probing technology. The challenge is to find a reasonable small set of probes for monitoring and make fault localization as fast as possible.

In [1], Kliger *et al* propose a coding approach to network event correlation. They view events as system-generated messages encoded in sets of alarms that the events cause, and the problem of correlation as decoding these alarms to identify the message. The coding technique proceeds in two phases, offline codebook selection and online decoding. In the codebook selection phase, a subset of alarms is selected as a codebook that optimally pinpoints problems and achieves a level of noise insensitivity. The small set of alarms is to be monitored constantly. When alarms are observed, the decoding phase starts. These alarms are analyzed to identify the problems that cause them. The coding approach reduces the complexity of real-time correlation analysis through preprocessing of the event knowledge model.

Manuscript received on December 11, 2004; revised on February 1, 2005. The author is with the Division of Mathematics and Computer Information Science, Mercy College, 555 Broadway, Dobbs Ferry, NY10522, USA (e-mail: zchen@mercy.edu).

More specifically, the codebook design problem is to find a minimal codebook such that the radius, defined as the one half of the minimal distance among pairs of codes, is bigger than a given level of distinction. In a deterministic model, the distance between two code vectors $a = (a_1, a_2, .., a_n)$ and $b = (b_1, b_2, ..., b_n)$ is defined as

$$d(a,b) = \sum d(a_k, b_k).$$

The Hamming distance is a special case. In a probabilistic model, a log-likelihood measure can be used, that is

$$d(a,b) = \sum \left| \lg\left(\frac{a_k}{b_k}\right) \right|$$

with $\lg\left(\frac{0}{0}\right) = 0$ and $\lg\left(\frac{0}{n}\right) = 1$ for $n \neq 0$.

The problem of decoding is to find for a given alarm vector a the problem codes P that minimize the correlation measure $\mu(a, p)$ for any $p \in P$. The correlation measure is an asymmetric mapping that distinguishes a lost symptom from a spurious one. In the deterministic model, a lost symptom correlation measure can be set as $\mu(0,1) = \alpha$ and a spurious symptom as $\mu(1,0) = \beta$. And the correlation measure $\mu(a, p)$ can be set as the form $\sum d(a_k, b_k)$ as before. In the probabilistic model, a correlation of occurrence $\mu(1, a) = |\lg(a)|$ while that of non-occurrence $\mu(0, a) = |\lg(1-a)|$. We can get the corresponding logarithmic form of correlation measure.

In [2,3], Brodie *et al* explore the probing technology to network fault localization. They use dependency matrix between network nodes and probes to describe diagnostic power and define diagnostic ability $H(P)$ of a set of probes P as

$$H(P) = \sum_{i=1}^{k} \frac{n_i}{n} \lg(n_i)$$

where n is the total number of nodes, k is the group numbers induced by P, that is, nodes in each group could not be distinguished further through P, and n_i is the number of nodes in group i. It is a form of conditional entropy in information theory [4] and can be used to define next most informative probe, the smaller the value of $H(P)$, the better the probe. Based on the diagnostic ability, they propose two algorithms, a subtractive approximation algorithm and additive algorithm to construct pre-planned locally optimal set of probes.

In the diagnostic phase, they extend dependency matrix with uncertainties into two layer noisy-AND Bayesian network that encodes probabilistic dependencies between the possible faults and symptoms. The first layer is of n un-

observed Boolean variables $X=(X_1, X_2,..., X_n)$ for nodes while the second layer is of m observed Boolean variables $T=(T_1, T_2,..., T_m)$ for probes. The joint probability $P(x,t)$ is defined as

$$P(x,t)=\prod_{i=1}^{n} P(x_i)\prod_{j=1}^{m} P(t_j \mid pa(t_j))$$

where $P(x_i)$ is the prior probability, $P(t_j \mid pa(t_j))$ is the conditional probability distribution of probe T_i given the node set $pa(t_j)$ to which T_i links directly. If l is the leak probability and q_i is the link probability, the lost and spurious probability are

$$P(t=1|x_1,...,x_n)=(1-l)\prod_{x_i=0}^{n} q_i$$

and

$$P(t=0|x_1=1,...,x_n=1)=1-l$$

The diagnosis task is to find the maximum probable explanation for a given probe outcome.

In [5], Steinder and Sethi map layered model for alarm correlation into brief networks and present three algorithms, bucket-tree elimination algorithm, polytrees algorithm and iterative most probable explanation algorithm, for end to end service failure diagnosis. More references about performance monitoring and fault localization including using neuronal network and support vector machine can be found in [6-10]

In this paper, we define a Probing Service (PS) as a web service that is able to run a specific test transaction, to record the test result, and to pass the result to a designated location called Probing Service Management Station (PSMS). Probing services are deployed in a Probing Service Outlet (PSO) in which a PS can be run under a regular schedule or invoked anytime. For example, an http request PS at certain PSO wraps a standard http request into a web service that sends out the http request, gets the response, records the response time and passes the result to a PSMS.

The generic architecture of probing service monitoring and diagnosis is depicted in Fig. 1 in which services component model is illustrated in Fig. 2.

The generic enterprise system services (ESS) model in Fig. 2 is a brief reference model [11]. It has application services such as portal server and messaging server. The main functionality of this layer is to provide application integration and user collaboration. The next layer is middleware services such as web server, application server, access manager and messaging queue. The main functionality of this layer is to provide business-ready infrastructure. The foundation layer is the operating system and network support.

In Fig. 1, we use 'services' block to denote an ESS that can be in one location or in geometrically dispersed locations. PS in PSO type I (PSO-I) is able to access ESS directly, usually inside the intranet while PS in PSO type II (PSO-II) accesses ESS through internet cloudy. PSO-I plays an administrative role while PSO-II mimics user perceived applications. PSs in both PSOs send response results to PSMS in which the response results are stored and analyzed. The PSMS has a database to store huge data, a data regulator to unify data presentation and to regulate time and distance measuring issues from different PSOs, a situation manager to cope with event uncertainty, a reference engine to analyze and diagnose events, an invoking channel to invoke a PS in local or remote PSO, and an alerting service to send alarms to system administrators.

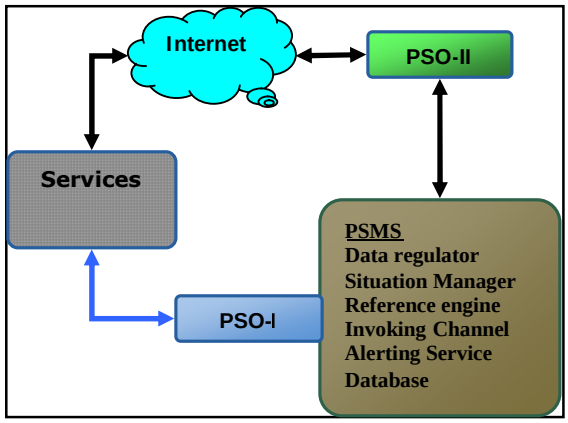

Figure 1: Generic Architectures of Probing Services

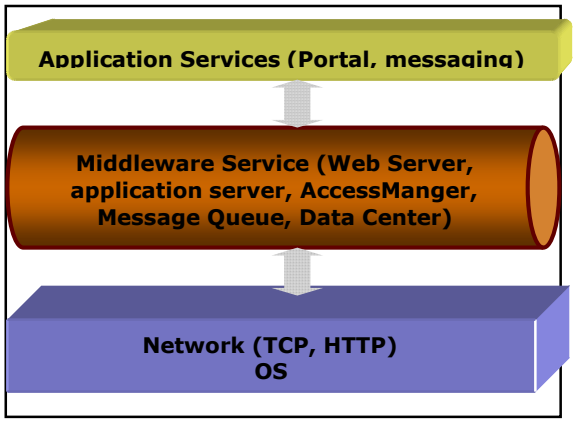

Figure 2: Generic Enterprise System Service Components.

We propose a three-stage approach called Proactive Probing and Probing on Demand, or simply pPOD approach. The first stage of the pPOD is to select PSOs and PSs. The goal is to monitor a set of components using as few PSs as possible. The information collected from these PSs does not serve the purpose to diagnose the nature of the problem later on. They simply monitor the health of the underlying components and report events. This de-coupling between monitoring and fault localization saves machine resources and traffic overload dramatically since in reality, majority time is spent in the monitoring phase. The second stage is offline rule development. The goal is to provide a strategy in case one PS or several PSs report events. It must consider all available PS pools when designing a faulty detection strategy. The last stage is to diagnose and localize faults using incremental probing on demand. It may start one or more PSs to get more information. The execution is based on the rules set by the second stage.

The paper is organized as follows. Section 2 presents several crucial concepts including a serviceable component system, probing services, events, and dependency matrix between a serviceable component system and a probing service set. Section 3 describes the pPOD approach and

explains the additive-pruning algorithm as an approximation to the exhausted algorithm. Section 4 discusses empirical experimental results. The final section is devoted to the discussion of the future work.

2. PROBING SERVICE AND SERVICEABLE COMPONENT

This section intends to clarify some main concepts used throughout the paper.

2.1 A Serviceable Component, Probing Service and Event

A serviceable component (SC) is a piece of hardware or software that is able to perform a specific task. For example, a cable is a hardware component that makes connection; an account entity bean in application server is a software component that manipulates and writes data into a relational database table; and a php is an application software component that is used for web interactive environment.

We may deal with a serviceable component that is actually a cluster of components among which we are unable to distinguish each individual component. For example, to handle high volume of traffic, some popular web sites use a proxy server followed a cluster of web servers. The proxy server allocates incoming requests to each individual web server based on some schedulers such as round robin or resource usage. In this situation, we could not distinguish each web server from application perspective. Therefore we treat the cluster of web servers as one serviceable component. Similarly, in a re-usable object case, a pool of objects may be treated as a component since there is no way to distinguish each object.

An event is an abnormal condition that a PS is able to detect. For example, timeout for an http request PS is an event. An event from a PS is only an indicator that some components along the path may not be in good condition. An event can be transient or consistent. Usually we ignore the transient events because they could not be re-produced. In our several month field tests, we found that from time to time an http request PS would report a burst of response time over threshold but the SCs the PS has passed through were actually in good health. They were transient events. A response time recording segment of an http request PS is shown in Fig. 3 in which the time frame is about 36 hours and the regular probing period is 5 minutes. The x-axis is time frame and y-axis is response time. The majority response time is around 100ms. Some transient timeout events were recorded.

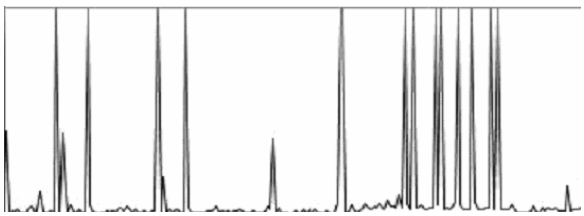

Figure 3: Response Time Sample from a PS

In this paper we do not use terms like symptoms and problems. The events are more or less symptoms while the SCs the problems.

A Probing Service (PS) is a web service that is able to run a specific test transaction, to record the test result, and to pass the result to a designated location called Probing Service Management Station (PSMS). In other words, a PS implements probing, logging, alerting and messaging interfaces, and supports direct interactions with other software applications using XML-based messages via Internet-based protocols. Its interfaces and binding are capable of being defined, described and discovered by XML artifacts and can be identified uniquely by a URI [12].

A PS can be run under a regular schedule or can be invoked anytime. The data format from any PS is in XML format that can be transformed into any data format by using XSL.

A PS can be deployed in any web environment that runs web service. The place is called Probing Service Outlet (PSO). If a PSO is within an intranet, it is termed as PSO-I while PSO-II for PSO outside an intranet. The reasons to have two types of PSO are due to the access rights and knowledge of targeted IT infrastructure topology. A PS in PSO-I can have administrative right to access a SC and is able to know the detailed topology of SCs. For instance, a lotus note PS in PSO-I is able to know the exact path to a lotus server, and can access lotus database and user setting directly. A PS in PSO-II usually has limited access right to fulfill its task. It could have an educated guess of the underlying topology. For example, through a different http URI and response time pattern, an http PS could guess that the request probably passes over an application server and/or a database server. Furthermore, a PS in PSO-I could get more useful information than that in PSO-II. For example, a traceroute PS in PSO-II usually could not get much information because such information is usually not accessible to outsiders.

A PS has to utilize and pass over several SCs in order to finish a specific task. For example, an http request inquiring the inventory of certain die-cast bike model might start from a browser in a local PC, use local or remote DNS, pass routers, and bridges to a remote computer, a web server and a presentation server, an application server with the inventory session bean and a database server. The success of such PS implies that all the SCs in between work properly.

Because all the SCs of a PS include a local computer to a target service, we further divide SC into two classes, Targeted Serviceable Component (TSC) and Necessary Serviceable Component (NSC). The goal of probing is to TSC. In the field test, through different Probing Service Outlet (PSO), we can assume that all NSCs work properly.

The Probing Service Management Station (PSMS) is a central processing server that comprises, as depicted in Fig. 1, a data regulator, a situation manager, a reference engine, an invoking channel, an alerting service and a database. Raw data from various PSs located at different PSOs are sent to the PSMS. Because these data come from different PSs and different locations, they might have different data representation format and have mingled time stamp. The data regulator transforms them into a standard data format using eXtensible Stylesheet Language Transformations (XSLT) [17] and adds proper time marker to these data. These data are stored to the database and at the same time are being processed by the situation manager. The situation manager makes a decision if a detected event should trigger diagnosis process or not. As we can see from Fig. 3 once a while a longer response time or even failure while the underlying SCs are actually in good health. The exact reason is hard to tell sometimes. We simply state they are due to measurement error and/or network

uncertainty. In such a transient situation, there is no need to trigger diagnosis process. Actually once we add a rule like "triggering diagnosis if the same event reported more than twice consecutively", we reduce unnecessary diagnosis about 90%. A situation manager is also useful in dealing with scheduled machine maintenance, notified machine updates, webpage changes, etc. The reference engine is used by PSMS in diagnosis. It can be deterministic rule engine or a magic statistical processing engine. We use simple rule engine at first. We plan to use combination of both deterministic and statistical mechanisms. During the diagnosis, one or more PSs may have to be invoked to get more information. The job is done by an invoking channel. Any diagnosis results will be sent to designated places like administrator email account or management console via the alerting service.

In short, the PSMS collects all the data from various PSs in different PSOs, monitors the health of the TCSs and makes diagnosis if events are detected. It can make on demand probing if extra information is needed during the diagnosis process. The advantages of PS-PSO-PSMS over a standalone software application like EPP [13] are the simplification of management, ease of scheduling, and short cycle of learning and development.

We should notice that not all the PS get responses instantly. Some more sophisticated PSs need correlated work to draw conclusions. For example, in a business process [14], an order transaction can have multiple players who input their outcomes in days or even months. The success or failure of these probe services depend on the outcome of each player. Our PS includes such orchestrated business processes.

We denote a PS and its covered SCs as a PS path, $C_{ps} = \{c_1, c_2, ..., c_{k_{ps}}\}$ where c_i is one of the SCs passed by the PS. All possible PS paths together form a PS impact space $S \equiv \{C_{ps}\}$.

2.2 Dependency Matrix and Serviceable Component System

We can associate every PS with a $1 \times N$ vector where N is the total number of SCs. The vector element value depends on whether the corresponding component is on the PS path or not. If it is, the value is set to 1, and otherwise to 0. We use V_{ps} to denote the vector that corresponds to a *ps* probe service.

If we line up every such vector together, row by row, we get a dependency matrix (DM) M that tells a relation between PS and SC. A matrix row-switching operation corresponds to PS exchanging while a column-switching one corresponds to SC swapping.

Experienced Information Technologies (IT) specialists usually are able to pin down that certain SCs have close relations among themselves and are independent of others. If we put these SCs together in a dependency matrix, we can get a block dependency matrix like the following.

$$M = \begin{pmatrix} M_1 & 0 & 0 & 0 \\ 0 & M_2 & 0 & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & 0 & M_k \end{pmatrix}$$

Each M_i is called a strong dependency matrix (SDM). We can deal with SDM one by one in a probing and diagnosis process. We call the group of SCs corresponding to M_i a Serviceable Component System (SCS), denoted as $C = \{c_1, c_2, ...c_N\}$. We will focus on an SCS for the rest of the paper.

As the complexity of IT systems grow tremendously, a machine-automated separation process is certainly desirable. It turns out that by using matrix notation and matrix row-switch and column-switch operations, a machine can transform a DM into a block DM easily. The process that separates a SC set into several disjoint sets is called a de-coupling process.

3. PROACTIVE PROBING AND PROBING ON DEMAND

In this section, we will formulate the problem and explain the approach to solve them. The approach is termed as Proactive Probing and Probing on Demand, simply, pPOD approach. The pPOD method has three stages, proactive probing in which a PS-PSO-PSMS system is selected, offline fault localization in which strategies for fault detection and localization are developed and some new PSs might be added to the available PS pools in order to increase the diagnosis power, and probing on demand fault diagnosis in which incremental results are fed into the PSMS reference manager for possible conclusions.

3.1 Proactive Probing

We denote the targeted serviceable components (TSCs) defined in Section 2.1 as

$$C_{TSC} = \{c_1, c_2, ..., c_n\}.$$

and probing services (PS) that pass over all the TSC as

$$P = \{p_1, p_2, ..., p_m\}.$$

The set C_{TSC} can be determined by a service level agreement and administrators while the PSs may be deployed in different PSOs. We do not label these PSs differently because we assume that the information passed into PSMS shows where they are located.

Usually we have one main PSO-I that may run in the same machine as PSMS, a PSO-II outside intranet and sometimes backup PSO-I or PSO-II for recovery and fault localization if needed. In practice, each PSO stores data locally. When there is an event, it reports to the PSMS for further instructions.

The existence of such P is obvious since we can use one PS to cover one TSC in theory. Put all such P together to form a probing service impact space $\tilde{P} = \{P\}$. Therefore the problem for PS selection is to find a $P^* \in \tilde{P}$ such that

$$|P^*| = \min_{P \in \tilde{P}} |P|. \quad (1)$$

If we use an exhaustive search algorithm, we can prove that the complexity is exponential. Intuitively from its dependency matrix, there are going to be at most 2^n PSs. For each $P \in \tilde{P}$, the number of PS, $|P|$, will be from 1 to 2^n. So the number of $|\tilde{P}|$ is around $\binom{1}{2^n} + \binom{2}{2^n} + ... + \binom{2^n}{2^n} \approx 2^{2^n}$. So its complexity will be exponential.

We can use the simple linear additive-pruning algorithm (APA) to get an approximation to P^*, as shown in Block 1.

```
Input: A TSC
Initialization: Set P empty
                Set DM (Dependency Matrix) null
Repeat the following until the TSC is empty
      Take out one SC from TSC;
      Make one PS that passes over the SC
      Add the PS to P
      Add the PS dependency row to the DM
      Take out SCs from TSC that the PS has covered
      If the row ▷ some rows in DM
          Remove these rows
          Removes the corresponding PSs from P
Output: P and DM
```

Block1: Additive Pruning Algorithm

Here we make a PS instead of going through the search of all available PSs. The $\triangleright$ operation of two vectors is defined as $v \triangleright u$ iff $v(i) \geq u(i)$ for each element. We can see that the complexity is $O(n)$.

As we see, both the solution P^* and its approximation solution are not unique. We can use locality concept [16] to measure how good a P is. Let r_i be the number of PSs in P passing over the *ith* SC in TSC. We define the Probing Service Locality as

$$L(P, C_{TSC}) = \frac{1}{m}\sqrt{\frac{1}{n}\sum_{i=1}^{n} r_i^2} \ .$$

The smaller the locality, the better the set P. For example, for $n = 20$ and $m = 3$, if all $r_i = 1$, the locality is 0.3333; if all $r_i = 2$, the locality is 0.6667; and if $r_o = 1, r_1 = 3$ and all the rest $r_i = 2$, the locality is 0.6749.

In practice, we take an approach called "Incubation" that gathers all available PSs to cover all SC in TSC and records the corresponding metrics such as response time. The process gives us good candidates for P as well as its metrics for alarming threshold determination. We can use some help tools such as web site logger (pageDetailer [15], for example) during the process.

The incubation process sometimes can reveal regular server maintenance schedule. This is useful in avoiding unnecessary diagnosis. The incubation process sometimes could even give us some indications of hidden or new SCs not listed in TSC originally. Because of the dynamic nature of application environment, some SCs such as a router, machine or application server may be added dynamically to a SCS without notifying its system administrators. Through various response time patterns (such as longer time consistently) and PSs (like traceroute), we are able to guess a hidden SCs and reverse the possible component layouts.

Lastly the incubation process can also help us identify new PS that needs to be added to the PS pool. For example, during a review session in the incubation process, we found that there was no PS to monitor the compliance of the policy: "no employees should open their schedules to the public in their lotus notes". Thus we added a new lotus PS. Depending on the size of the TCS, the incubation process could take as long as several weeks in order to establish a baseline for PS selection and reference measurement metrics.

3.2 Offline Analysis

Assume that we have selected a P that passes over all SC in C_{TSC} and has the NSC (see Section 2.1), denoted as

$$C_{NSC} = \{d_1, d_2, ... d_k\}.$$

The proactive probing process actually monitors both C_{TSC} and C_{NSC}, or $C = C_{TSC} \bigcup C_{NSC}$. For each PS, we can write a causality line as the order of the PS passing over, symbolically as

$$p_i \propto d_{i_1} \rightarrow d_{i_2} \mapsto ... \mapsto d_{i_\alpha} \rightarrow c_{i_1} \mapsto c_{i_2} \mapsto ... \mapsto c_{i_\beta}$$

where all d_{i_k} belong to C_{TSC} and all c_{i_k} to C_{TSC}. The operation $c_1 \mapsto c_2$ means before passing over c_2, p_i has to pass over c_1 first. If p_i returns a successful response, all the SCs it passed over are in good health. Otherwise, they are all possible candidates for further diagnosis. We can reasonably assume that $\{d_{i_k}\}_1^{i_\alpha}$ in C_{NSC} are in good health. Our job is to focus on $\{c_{i_k}\}_1^{i_\beta}$. The problem now is to design a strategy that uses as few available PSs as possible to decide possible faulty SCs in $\{c_{i_k}\}_1^{i_\beta}$.

The approach we use is similar to a binary search algorithm. It tries to find an available PS that stops right at the middle of the causality line or close to the middle of the causality line. Because we are doing offline proactive analysis, we have time even to develop a PS that ends around the middle of the causality line if there is no suitable PS. The algorithm is given in block 2 if we assume that there is only one faulty SC in the causality line.

```
Input: A main causality line MCL= {c_{i_k}}_1^{i_β}
Initialization: Set the possible faulty set F = {c_{i_k}}_1^{i_β}
Repeat the following until the MCL is empty
or there is no PS that could stop between the MCL
      Find a PS that ends c_j , around the middle of MCL,
      If the result from the PS is good
        Set MCL as c_{j+1} to the end
        Remove c from F all the way to c_j
      Else
        Set MCL as from the beginning to c_{j-1}
        Remove c from F from c_j to the end
Output: F
```

Block2: Diagnosis Algorithm

The complexity of the algorithm is close to $O(\log(n))$. The output F can have unique SC or several SCs. The reason to have several SCs is because there might be no suitable PS that can penetrate the causality segment by the SCs.

If we remove the assumption of only one faulty SC, the algorithm will be more complicated. We can not "remove c from F from c_j to the end" in the above algorithm if the result from the PS is not good. We might need a PS from different

PSO to bypass the path from the beginning to c_j. But this is not always possible in the field test.

3.3 Probing on Demand

An event reported to PSMS by a PS does not necessarily trigger a diagnosis process.

An event can be transient. In our field test, we observed many transient events among them some were very regular like a burst every two hours. They simply come and go without revealing what is happening behind.

An event can be consistent but not necessarily caused by a faulty SC. For example, regular server maintenance could cause a consistent event. Actually our probing to a Lotus notes server disclosed its maintenance schedule, starting every Thursday 5 am and lasted about average 7 minutes. Certainly 7 minutes is not very accurate number because of the various natures of the maintenance and the gap between two consecutive probings.

A situation manager in PSMS takes care of the uncertainty and regular machine and server maintenance. It can eliminate many unnecessary diagnoses. One simple way to deal with uncertainty is to ask the same PS who has reported an event to perform the task again to see if the event is consistent or not, or ask other PLO to perform the same task.

Now, for a consistent event, the diagnosis basically follows the rules developed during the offline analysis. Based on the return value, it decides the next move.

Unlike event correlation [1] and intelligent probing [2], the diagnosis starts without information provided by other PSs. Whenever an event is reported, the proactive probing process is stopped.

4. Examples and Results

We investigate empirically the relation between PS size and C_{TSC} size, and the difference between the two algorithms, Exhausted Search Algorithm (ESA) and the Additive-Pruning Algorithm (APA). Moreover, we also investigate the average diagnosis execution time units based on the PS set obtained via APA.

For each number $n = |C_{TSC}|$, we generate n PSs that cover all the SCs in C_{TSC}. The number of SC passed over by each PS is uniformly distributed between 1 and 60% of n. The algorithms described in section 3.1 are executed. The process is repeated fifty times for each SC size n. Fig. 4 shows the results. The x-axis is C_{TSC} size (n) and the y-axis is PS set size (m). The dotted lines are the results from ESA and the solid lines from APA. The standard deviation bars are also drawn along with the average results.

From Fig. 4, we see that the number of PSs needed to monitor the C_{TSC} is quite small. For example, when the size of C_{TSC} is 150, the number of PS needed is only 6. Moreover, the difference between ESA and APA is around two PSs. We can thus conclude that APA is a good approximation to ESA.

Now we use the PS set obtained from APA as a base to do offline diagnosis analysis discussed in Section 3.2. Fig. 5 shows the average diagnosis execution time unit that is equivalently the number of PS needed to diagnose a faulty SC. Here we assume that there is only one faulty SC. The x-axis is the C_{TSC} size and the y-axis is the number of PS needed in order to be able to localize a faulty SC. We also draw the function of $\log_2(C_{TSC})$ as a dashed line. It serves as a reference line. As we can see that all the numbers of PS needed to diagnose are below the line. That means that the number is smaller than the logarithm of C_{TSC} size. The difference is about two execution time units.

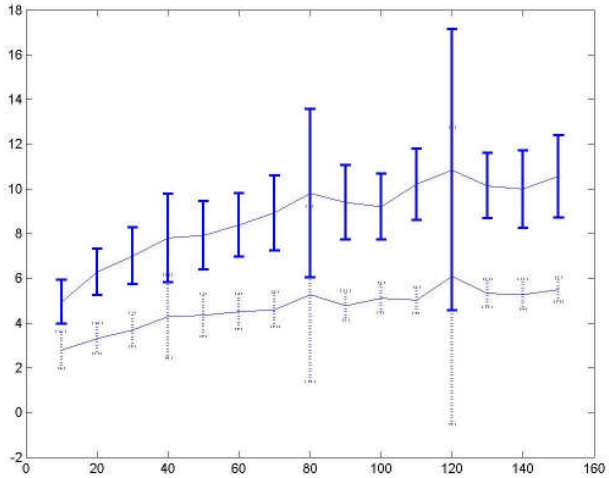

Figure 4: Computing PS set size via ESA and APA

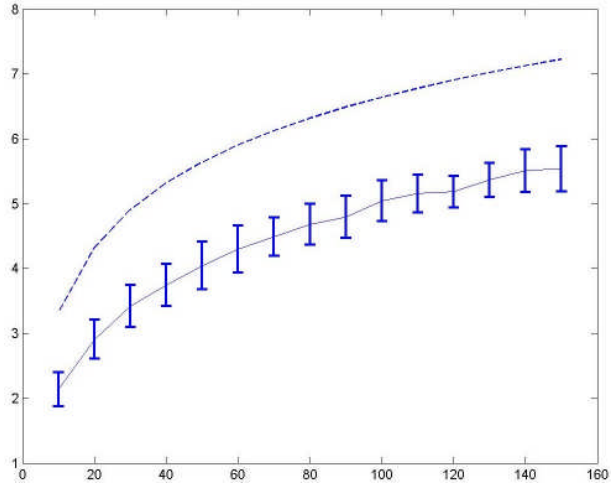

Figure 5: Average Diagnosis Execution Time Units

We also used the PS-PSO-PSMS system for a several month field test. The configuration included three lotus notes servers (one lotus notes server was within the intranet, one lotus was also within the intranet but severed as a backup, the third one was outside the intranet) and portal servers as interface between lotus notes servers and web access, their basic operations such as open, close and send a file, a document retrieve and edit. We used pageDetailer with SSL capability to get all possible http requests. The probing period was set as five minutes. We revealed the regular machine maintenance schedules, periodical transient timeout events with period approximately two hours (which we did not find reasonable explanations such as caching invalidation), one

unavailable document event and two downtime events of the outside Lotus notes server that lasted about one half hour. Our results matched the administrators' working log quite well.

5. Discussions

In this paper, we have proposed an Additive Pruning Algorithm for proactive probing and a diagnosis algorithm for fault localization. It reduces the probe service numbers dramatically while at the same time increases the effectiveness of diagnosis since the process is dealt with only a subset of the whole C_{TSC} .

Our ongoing work includes applying these algorithms to more sophisticated IT systems; handling dynamic nature of SCS by allowing components to be added or removed dynamically; and revealing more quantity measurement to understand the nature of different probe services. We also like to predict events from user-perceived performance pattern change. The work is more based on our working experience. We need to perform in depth mathematical analysis of the proposed method and its effectiveness.

REFERENCES

[1] Kliger, S., Yemini, S., Yemini, Y., Ohsie, D. and Stolfo. S., "A coding approach to event correlation," in *Proceedings of the Fourth International Symposium on Integrated Network Management*, 1995, pp. 266 – 277.

[2] Brodie, M., Rish, I., and Ma, S., "Intelligent probing: A Cost-Efficient Approach to Fault Diagnosis in Computer Networks," *IBM Systems Journal*, Vol 41, No. 3., 2002, pp. 372-385

[3] Rish, I., Brodie, M., Odintsova, N., Ma, S. and Grabarnik, G., "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, Vol1, pp 133-146. See more discussions at http://www.research.ibm.com/people/r/rish/

[4] Cover, T. and Thomas. J., "Elements of Information Theory," New York, John Wiley & Sons, 1991.

[5] Steinder, M., and Sethi, A., "End-to-end Service Failure Diagnosis Using Belief Networks," in *Proc. NOMS-2002, 8th International IFIP/IEEE Symposium on Network Operations and Management*, Florence, Italy, pp. 375-390, April 2002

[6] Steindera, M., and Sethi, A., "A survey of fault localization techniques in computer networks," *Science of Computer Programming*, Vol 53, 2004, pp. 165–194.

[7] Hood, C., and Ji, C., "Proactive network fault detection". *In Proc of INFOCOM '97, Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, 1997, pp. 1147-1156

[8] Kandula, S., Shaikh, A. and Nahum, E., "Integrated Network Performance Diagnostics," *NYMAN Workshop*, New York, NY, September 2002. An Electronic copy can be obtained at http://www.nyman-workshop.org/2002/papers/s2343.pdf

[9] Papavassiliou, S., and Pace M., "From service configuration through performance monitoring to fault detection: implementing an integrated and automated network maintenance platform for enhancing wide area transaction access services", *International Journal of Network Management*, Vol. 10, No 5, 2000, pp. 241 – 259.

[10] Samanta, B., Al-Balushi, K., and Al-Araimi, S., "Artificial neural networks and support vector machines with genetic algorithm for bearing fault detection", *Engineering Applications of Artificial Intelligence*, Vol 16, No 7-8, 2003, pp.657-665

[11] Sun Java Enterprise System 2005Q1 Technical Overview, http://docs.sun.com/source/819-0061/index.html, Chapter 2, Java Enterprise System Solution Architectures.

[12] Web Services Description Requirements, http://www.w3.org/TR/ws-desc-reqs/, Section 2, Definitions.

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

[14] Business Process Execution Language for Web Services (BPEL4WS), Version 1.1, May 2003, Electronic version can be obtained from ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

[15] pageDetailer, http://www.research.ibm.com/pagedetailer/

[16] Salisbury, C., Chen, Z. and Melhem, R., "Modeling Communication Locality in Multiprocessors," *J. Parallel and Distributed Computing*, Vol 56, pp. 71-98, 1999

[17] http://www.w3.org/TR/xslt

Zhixiong Chen received his PhD degree in Mathematics and Master's degree in Computer Science from the University of Pittsburgh in 1997. He received his Master and Bachelor degrees in Mathematics from Shanghai Jiao Tong University in 1989 and 1996, respectively. Dr Chen is an Associate Professor in the Division of Mathematics and Computer Information Science at Mercy College, Dobbs Ferry, New York. He worked over six years in IBM Transarc lab and IBM T J Watson Research Center doing distributed application research, design and implementation prior to his present position. His research interests include autonomic computing such as multi-layer application measuring, monitoring and fault localization using various analytic, fuzzy and statistical methods, security, parallel computing and neuronal modeling. He had published numerous journal papers, conference papers and technical reports. He is a member of IEEE and MAA.