ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2, Issue 1, January 2013

Fault identification and diagnosis using greedy Back Tracking approach in large-scale network

U.Sridhar[1], Dr.G.Gunasekaran[2]

Abstract - **In this paper, we discuss the problems in large scale networks failure information. For this We propose a fault identification system initiate by an application when the application identify the incidence of a failure, in order to realize supervision systems that automatically find the foundation of a failure. In existing detection systems, there are three problems for constructing self managing applications: i) the detection results are not sent to the applications, ii) they cannot identify the source failure from all of the detected failures, and iii) configuring the detection system for networked system is hard work.**

To overcoming these problems, the proposed system takes three approaches: i) the system receives failure information from an application and returns a result set to the application, ii) the system identifies the source failure using greedy technique, and iii) the system obtains information of the monitored system from a database. The error relationship is expressed by a tree. This tree is called greedy error tree. The database provides information which is system entities such as hardware devices, software object, and network topology. When the proposed system starts looking for the source of a failure, causal relations from a greedy error relation tree are referred to, and the correspondence of error definitions and actual objects is derived using the database. We show the design of the identification operation activated by the failure information and the architecture of the proposed system.

Index Terms: fault detection, fault diagnosis, error relationship tree, greedy approach, Backtracking, application-level failures

I. INTRODUCTION

In a large-scale networked system, many components, such as computer hardware devices and software services, have been installed. It takes time to determine and locate the issues of failures because of the difficulty of dependencies in such components. As networked systems grow in size and complexity, it becomes increasingly impractical to observe each component manually for sources of failures. Manual diagnosis does not scale. An automated approach is essential for autonomic computing [12].

First of all, let us define the terms failure, error, and fault based on definitions in another paper [3]; a failure is an event that occurs when the delivered service deviates from correct service; an error occurs when a component providing a capability or a service does not provide an expected response; and a fault is the cause of an error. That is, a fault causes an error. The error causes subsequent errors, and then a failure occurs. An application program can receive a failure report.

U.Sridhar, Research Scholar, CMJ University,. Meghalaya, India, Mob:7871338432.

Dr.G.Gunasekaran,, Principal, Meenakshi College of Engineering, West K.K.Nagar, Chennai, Tamilnadu, Mob: 9444177718

Fault detection approaches for networked systems monitor the status of processes and hardware devices periodically [1, 8]. In such a fault detection system, relationships between monitoring operations and target objects are defined by the system configuration. The system monitors a target using this relationship, and it finds the fault.

There are three problems in existing fault detection systems: i) the detection system is currently independent in applications; ii) identification of failure source, and iii) the complexity of the monitored system configuration. Since the detection system is independently monitoring computer resources, such as IBM Tivoli Monitoring [8] and Nagios [1], an application cannot obtain information on what kind of fault has happened. Thus, the application cannot perform self-management against external errors. In the issue of the identification of a failure source, existing systems [1, 8] cannot identify the source of failures, though they can find some faults.

For example, suppose that a fault detection system monitors certain hosts over a network switch. If the network switch is broken, the system might report two types of problems: one is the network switch fault and the other is faults in all of the services in the hosts over the switch. Though the latter faults are caused by the network switch fault, the system cannot identify this relationship. Another issue, the complexity of the monitored system configuration, is due to the fact that it is complex work to define and maintains the relationships between monitoring operations and target objects in a large networked system. For example, Nagios [1] uses relationships among monitored objects based on the network topology in order to determine the network reachability of the monitored objects. If the network topology is changed, it is necessary to redefine the relations.

In this paper, we propose a new fault detection & identification system activated by an application program, which overcomes problems associated with these problems. Two key features are introduced as follows. One is a relationship tree that is introduced in order to help find the source of an error. This tree is called an greedy error tree. Because an error is caused by other errors, each of which is caused by yet other errors, and so on, error relationships are represented by a tree structure. The error relationship tree is independent from the actual hardware configuration, e.g., network topology, servers, disks, CPUs, network switches, and so on.

Another feature is that all configuration elements of the monitored system are specified based on an object-oriented database, CIM [6] defined by the DMTF organization. In addition to the DMTF definition, a fault and its resolver are also defined based on CIM. The detection engine detects the source of a failure reported by an application by traversing an error relationship tree using the database.

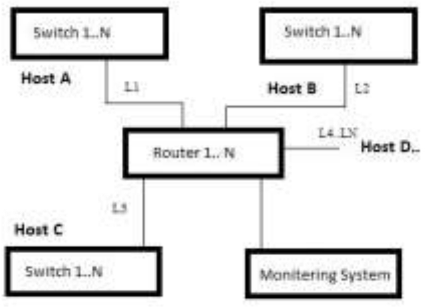

Figure 1: Network Topology

II. ISSUES

The first issue in Section 1 is a straightforward issue. The last two issues addressed in Section 1 are described in detail in this section. An example of the issue of the identification of a failure source is shown using a network topology in Figure 1. We assume that a detection system monitors remote hosts and a network router in Host A. When the network router is broken, most existing detection systems will detect several failures; e.g., a network router fault, communication failures involving Host B and Host C and so on, and failures due to unavailability of services provided in Host B and Host C. Most of the failures detected are useless because the user wants to know about the network router fault, the original cause of the failures, but does not want to receive secondary failure reports caused subsequently by the original source.

The useless announcement could be reduced by taking the network communication status into consideration. It is realized by Nagios [1]. Nagios uses network reachability tree whose root node is the host running the Nagios server. Each node has a parent which is one hop closer to the monitoring server. A monitored object, such as a server activity, is also specified in this tree. For example, in Figure 1, Host A is the root and Router is a child node of the root. Both Host B and Host C are child nodes of Router. When the Nagios system receives notification of several failures, it tries to find the root of these failures using this tree. It is difficult to specify a large networked system because the tree must be generated manually depending on the network topology. Moreover, when the topology is changed, it is difficult to adapt to the changes without exception or discrepancy.

Next, the second issue, the complexity of the monitored system configuration, will be discussed. Generally speaking, the administrator works as follows in order to monitor a resource, such as a hardware device or a software service:
i) She/he decides the types of faults which the detection system must detect, and defines the detection method;
ii) A device/service and its detection method are associated so that the system monitors the resource automatically; and
iii) Some causality relationships are programmed to detect compound faults. Because detection methods and resources are tightly coupled in existing fault detection systems, the above process must be performed when a new device/service is installed. For example, consider a new computer installation. The administrator must re-initialize the detection system in order to monitor local devices such as CPUs, memory, disk, fans, and so on. Then the action to be carried out upon notification of the monitoring results must be decided for each device. The network connection and services are global information
in the sense that the malfunction of such a resource causes a system-wide error. Thus, the causality relationships are greatly changed.

III. PROPOSED FAULT DETECTION SYSTEM

To defeat the problems addressed in the previous section, we introduce i) a new detection & identification system activated by an application, ii) the greedy error tree, and iii) the device and network database described in this paper. Unlike other fault detection systems, the proposed detection system is activated by an application which receives a failure notification reported by an operating system or another application. This device enables an application to choose an alternative action according to the failure type.

In the proposed system, an application sends the failure information, including some related parameters and the detection system, then, reports the source of the failure that contains the failed device/service name. In this example, the application receives the failure of Host C as HostUnreachable. The other failures caused by this source failure, such as PinterNotFound, File Not Found are not reported. The application prompts the user to recover the host.

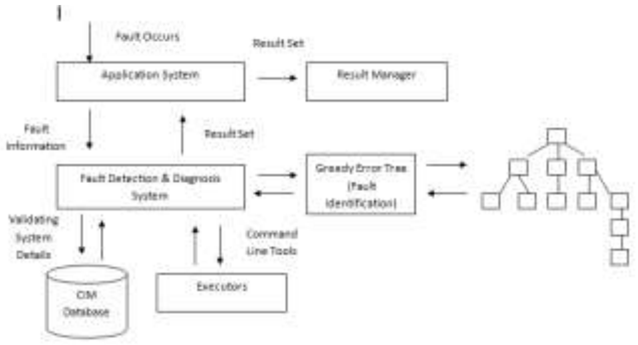

Figure2: Architecture of detection & identification system

The architecture of the detection & identification system is shown in Figure 2. The detection & Diagnosis system contains six components, as follows: Application Manger receives the Failure information and is translated into internal parameters by the Translator. Fault Detection & Identification Manager the Fault Detection Manager finds the source of the failure using an greedy error tree. The tree is instantiated with the execution environment defined by the database. Because an error is caused by other errors, each of which is caused by yet other errors, and so on, this error-to-error relationship is represented by a tree.

The Fault Detection Manager traverses the greedy error tree and invokes requests to the Database Adapter for querying information on monitored objects. It invokes the Executor to perform a diagnosis according to information on the tree and monitored objects. Based on the results, the source of the failure is determined and sent to the Resultant Manager. Database Adapter the Database Adapter invokes a query, requested by the Fault Detection Manager, and sends it to the CIM database, Executor the Command-Tool Executor executes a program used for diagnosis and returns its results to the Fault Detection Manager. The diagnosis results are classified into three types: 1) no error, 2) some errors, and 3) execution failed. The result is classified as the third type if the diagnosis program does not return a result within a certain time. Resultant Manager the Resultant Manager receives information on the source of failure, translates it to application readable information, and then sends it to the application.

In the rest of this section, the greedy error tree and the device and network database are introduced.

In order to explain the semantics of error relationship trees, let us assume the following environment in the rest of this paper: i) the execution environment depicted in Figure 1, ii) Host A is in Network Communicated to the Host B Server, and iii) when an application fails, it is because a remote shell execution command sent to Host B failed.

A. Fault Identification

In this paper we used the technique backtracking for finding the optimal solution for the fault identification. Back tracking is a modified depth-first search of a tree consisting of all of the possible sequences that can be constructed from the problem set. In backtracking we perform a depth-first traversal of this tree until we reach a solution node that is non-viable or non-promising, at which point we "prune the sub tree" rooted at this node, and continue the depth-first traversal of the tree. If you use the normal Binary tree it will travel up to the end of the child. But when you use the Backtracking It will stop the process whenever the solution occurred.

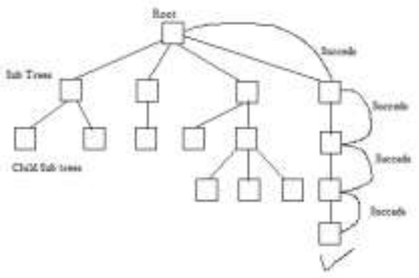

Figure 3: Sample Backtracking Tree Structure

B. Algorithm to find the Fault Information

Algorithm 1: Fault Detection Algorithm using Backtracking method

```
public static void faultidentification( node v)
{
    if (promising(v) )
        if (there is a solution at v)
    report the solution;
        else {
     for (each child u of v)
     faultidentification(u);
}
```

From the above algorithm we used the backtracking technique. First the identification starts from the root node of the problem and then moves down to the sub trees to find the exact solution in a step by step manner. Hence we can find a optimal solution of the problem. If any one of the condition fails, it stops the process and reverts back with the solution.

Running time efficiency of the dynamic algorithm (Backtracking) tree is low compare to the Normal Binary tree. For an example, for n objects, there are 2^n possible solution sets -- the i^{th} object is either in or out of the solution set. The state space, therefore, is represented as a complete binary tree with 2^n leaves. Such a tree will have a total of $2^{n+1} - 1$ nodes. The backtracking algorithm, therefore, has a worst-case bound that is $O(2^n)$. With pruning the actual number of nodes "visited" by the algorithm is much less.

C. Device and Network Database

The complexity of the monitored system configuration is lessened by introducing an object-oriented database, CIM (Common Information Model) [6] defined by the DMTF organization. CIM is a conceptual information model for information system entities, such as hardware devices, services, and networks. Applications obtain device information via the common CIM interface, and are able to solve dependencies in connections of network nodes, such as a next-hop node by querying to this database. Table 1 shows an example of the Device Information. Note that for simplicity, an object name does not follow the CIM convention in this paper. Two hosts, one Host B, and one network router are expressed in this figure. All hosts and the Host B are connected by a network router. This database provides information about the monitored environment to the detection system, such as a property of an object, which may be the name of a service, an IP address, routing information, and so on. Another example of a property is an association between objects, such as a computer and a service running on the computer. Table 1 is the Device Configured Information for the example Network shown in Figure 1. In this figure, the Connection table expresses the connection between the Node and Link objects where Node is Router, Host A, Host B, or Host C, Host D..N, and a Link object represents the connection between Nodes. For example, Router and Host A are connected by L1. The Routing table expresses the network routing information which consists of hostname, IP, MASK, and Link columns.

Table 1: Device Configure Information

Link	Node	Host Name	IP Address	Mask Address	Link
L1	Router	Host A	192.168.10.xxx	255.255.255.0	L1
L2	Router	Host B	192.168.11.xxx	255.255.255.0	L2
L3	Router	Host C	192.168.12.xxx	255.255.255.0	L3
L4	Router	Host D	192.168.13.xxx	255.255.255.0	L4
...	..	..	..	..	..
LN	Router	Host D	192.168.50.xxx	255.255.255.0	LN

IV. Implementation

A prototype of the proposed detection system has been implemented in Linux. As shown in Section 3, the proposed fault detection system consists of the Fault detection engine, detection tools, greedy error tree database, and CIM database. The Fault detection engine is written in C. Most of the detection tools are shell scripts using Linux commands such as ping and ifconfig. To monitor hardware devices, IPMI [10] and SNMP [2] protocols are used.

The CIM database is realized using an SQL server. Because the CIM database is an object-oriented database, it cannot be handled directly by the SQL server. In order to express objects, each table representing objects has an internal column, UID (unique identifier). An object is referred to using UID. A compiler, by means of which CIM objects are compiled into SQL tables, has been implemented.

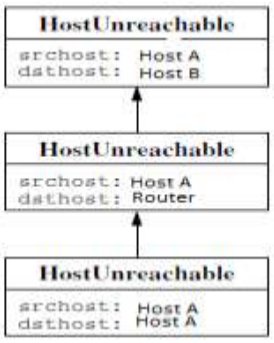

Figure 4: Communication error between Host A & Host B

As an example of the detection sequence, the traversal of HostUnreachable node is taken into consideration. Suppose that the fault detection manager instantiates the greedy error tree using the HostUnreachable error node under the database configuration shown in Table1.

The HostUnreachable node represents a communication error from srchost to dsthost, shown in Figure 4. Under this configuration, the top node is an instance of the HostUnreachable with srchost as Host A and dsthost as Host B. During the instantiation phase, the instantiation definition is applied to generate the tree.

Since the srchost and dsthost are different, another HostUnreachable node is instantiated as follows. The HostUnreachable node contains the database query statement in the instant ion definition. This query returns the host name, Router, which is one hop closer to Host A in this case. Using this result, the successor node is generated, and the instantiation definition of the new node is evaluated. This kind of instantiation is continued until the srchost and dsthost are the same value in a node. As a result, the instantiated tree is obtained shown in Figure 4.

The fault detection manager traverses the tree by back Tracking (DFS) search. That is, the program defined in the script part in each node is evaluated in the order from the bottom node. In this example, the diagnosis tag shown in Figure 5 is evaluated using the CIM database. As a result, the diagnosis system command is carried out and it is invoked by the Executor. If the command reports no error, the node is removed. If the command reports the error status, the node remains. After all nodes are evaluated, the evaluated tree contains the source errors.

```
<error name="HostUnreachable">
        <param name="srchost"/>
        <param name="dsthost"/>
        <instantiation>
         <if><test> <eq>
                <var name="dsthost"/>
                <var name="srchost"/>
        </eq> </test>
        <then><result value="NoError"/></then>
        <else> <source name="HostUnreachable">
        <arg name="srchost">
                <var name="srchost"/> </arg>
        <arg name="dsthost">
        <cim query="(query shown in Fig.8)"/>
        </arg> </source> </else>
        </if> </instantiation>
        <script>
                <diagnosis
                        op="(query to get an operation)">
                        <arg name="host">
                                <cim query="(query to
                        get the host
                                object identified by
                        srchost name)"/>
                        </arg> </diagnosis>
        </script>
</error>
```

Figure 5: Diagnosis Result

V. Related Work

The error relationship tree is similar to Fault Tree Analysis (FTA) [14] which is an analysis method used in reliability engineering. FTA is used in tracing a low-level fault contributing to major faults or undesired events, and used in calculating a probability of the occurrence major faults or undesired events based on the basic probability of the occurrence of a low level fault. There are logical gates such as .AND. and .OR. to construct fault components from lower faults. .AND. means that a next level fault occurs if all of the lower faults occur. .OR. means that a next level fault occurs if any of lower faults occur. In the model based on FTA, errors which occur when faults occur at the same time and errors which occur only when one of the faults occurs are represented.

Unlike FTA, the error relation tree proposed in this paper does not have .AND. relation because this relation does not need fault detection, as explained below. In the case of errors derived from an .AND. gate at FTA, all faults are sources of the gate, so that this case produces the same result as monitoring all faults in the detection system.

Chen et al. [5] use decision trees for diagnosing failures in a large system. The system monitors the client-server activities so that each request is recorded and applied to aid in learning by a decision tree. Then the decision tree is applied to each recorded log for diagnosing the problem. The system can detect failures of computer hosts or services where requests are recorded. The system cannot detect a fault in a device which does not appear in any request, such as a network switch.

Brodie et al. [4] use symptoms of software problems. A few problems are very common in many large software products and are major cost components of product support. The symptoms of such recurring problems are compared against the database of symptoms for diagnosis. They use the program call-stack as a symptom. The resolution of the symptom is generated with regard to the call-stack. Although other types of symptoms are supported by their architecture, their approach is of no use in diagnosing hardware devices.

The IBM Tivoli Monitoring System [8] detects bottlenecks and potential problems by periodic monitoring. In this system, the monitoring engine performs monitoring operations on each host. The logical model, which contains periods for collecting device data, monitored devices, thresholds, and so on, is used by the engine to collect numeric results for devices. The system reports a fault occurrence when the monitored resource exceeds the threshold or suffers a malfunction. The system is configured by deploying a profile which is a container for resource models using the Tivoli Management Framework [7]. This system reports a fault occurrence by correlating faults. For example, the indication that CPU time usage is high and one or more processes use too high a percentage of CPU time are reported when two indications occur. One indication is that a process uses an excessive percentage of CPU time. Another indication is that only a few processes use a high percentage of CPU time.

The Tivoli system is designed for monitoring hardware devices and the software objects of each host. It takes quite an effort to expand its functionality, such as the detection of network communication problems. Substantial experience in Java is required for the functionality expansion [9]. Nagios [1] monitors servers, workstations, switches, and services. To configure this system, a host name, an IP address, an inspection command, and so on, are defined for each host and service. Nagios can determine whether the remote host is down, or the host is unreachable. Whenever an inspection command results in a non-OK state, Nagios will attempt to check and see if the host, where the service is running, is alive.

Nagios uses a tree for determining reachability. When a check for a remote host returns a non-OK state, Nagios will check the parent host until the parent host check results in an OK state. This tree is constructed by describing the option denoting the parent host in each host definition. The configuration of this option is complex work when it has to conform to a large network system. The option is defined in each host definition, so when the network configuration is changed, it is a very complex task to reconstruct the dependency.

Katzela et al. [11] and Steinder et al. [13] localize the original cause of a failure by probabilistic approach. This approach can identify the original cause of a failure faster than the approach checking all of components in the network system. But, there is a problem of false positive in the probabilistic approach. The proposed detection system checks specific components concerned with the information of a failure. There is no false positive because of checking the components.

VI. CONCLUSION

We have proposed a Fault detection & identification system that is initiated by the failure information reported from an application, and that returns the source of the failure to the application. Because the existing fault detection systems do not provide interfaces for applications, such an application cannot obtain information on what fault happens. Thus, an application cannot perform the proper workaround for the fault under existing fault detection systems. Unlike such systems, the proposed fault detection system reports the error source to the application so that the application can manage its behavior.

The proposed system also overcomes two other issues in existing fault detection systems: i) identification of failure source; and ii) the complexity of the monitored system configuration. The greedy error tree has been proposed for the issue involving the identification of the failure source. Each tree represents an error and its possible error sources. This tree is independent of the actual hardware configuration, and thus it is applicable for any environment. In order to relax the second issue, a CIM (Common Information Model) database is employed to describe hardware devices, applications, and network topology. The detection system detects the source of a failure reported by an application by traversing a greedy error tree using the database.

REFERENCES

[1] Nagios. http://www.nagios.org/.
[2] SNMP. http://www.ietf.org/rfc/rfc1157.txt.
[3] A. Avi_zienis, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing,1(1):11.33, Jan.-March. 2004.
[4] M. Brodie, S. Ma, G. Lohman, L. Mignet, M. Wilding, J. Champlin, and P. Sohn. Quickly Finding Known Software Problems via Automated Symptom Matching. In International Conference on Autonomic Computing, 2005.
[5] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure Diagnosis Using Decision Trees. In International Conference on Autonomic Computing, 2004.
[6] DMTF. Common Information Model. http://www.dmtf.org/standards/cim/.
[7] IBM. IBM Tivoli Management Framework. http://www-306.ibm.com/software/tivoli/products/mgt-framework/.
[8] IBM. IBM Tivoli Monitoring. http://www-306.ibm. com/software/tivoli/products/monitor/.
[9] IBM. IBM Tivoli Monitoring Version 5.1.1 Creating Resource Models and Providers, Aug 2003. Redbooks.
[10] Intel, Hewlett-Packard, NEC, and Dell. Intelligent Platform Management Interface Speci_cation, Second Generation v2.0, February 2004.
[11] I. Katzela and M. Schwartz. Schemes for Fault Identi_cation in Communication Network. IEEE Transactions on Networking, 3(6):753.764, Dec 1995.
[12] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. IEEE Computer, 36(1):41.50, Jan 2003.
[13] M. Steinder and A. S. Sethi. Probablistic Fault Localization in Communication Systems Using Belief Networks. IEEE Transactions on Networking, 12(5):809.822, Oct 2004.
[14] W. E. Vesery, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. Fault Tree Handbook. U.S. Nuclear Regulatory Commission, Washington D.C., Jan 1981. NUREG-0492.

U.Sridhar received the B.Sc. degree in Computer Science from Annamalai University,tamilnadu,india, in 2002, and the M.Sc. degree in Information technology from Madras University, in 2004, and the M.Tech. degree in Information technology from Sathyabama University, in 2007 and is currently pursuing the Ph.D. degree in Computer Science & Engineering at the CMJ University, Meghalaya, India. His research interests are in the **areas** of Fault identification in Dynamic System and design and analysis of network algorithms and network protocols and defense mechanisms against malicious traffic and Database maintenance.

Dr.G.Gunasekaran received the B.E. degree in Computer Science from Madurai Kamarajar University,tamilnadu, india, in 1989, and the M.E. degree in Computer Science from Jadavpur University, in 2001, and the Ph.D. degree in Data Mining at the Jadhavpur University, Kolkata, India in 2009. His interests include network coding, network measurements and security, and Data Mining.