

PDA: A Tool for Automated Problem Determination

Hai Huang, Raymond Jennings III, Yaoping Ruan, Ramendra Sahoo, Sambit Sahu, and Anees Shaikh – IBM T. J. Watson Research Center

ABSTRACT

Problem determination remains one of the most expensive and time-consuming functions in system management due to the difficulty in automating what is essentially a highly experience-dependent task. In this paper we study the characteristics of problem tickets in an enterprise IT infrastructure and observe that most of the tickets come from very few products and modules, and OS problems present higher resolving duration. We propose PDA, a problem management tool that provides automated problem diagnosis capabilities to assist system administrators in solving real-world problems more efficiently. PDA uses a two-level approach of proactive, high-level system health checks, coupled with rule-based “drill-down” probing to automatically collect detailed information related to the problem. Our tool allows system administrators to author and customize probes and rules accordingly and share across the organization. We illustrate the usage and benefits of PDA with a number of UNIX problem scenarios that show PDA is able to quickly collect key information through its rules to aid in problem determination.

Introduction

Computer system administrators (SAs) play a number of important roles in managing enterprise IT infrastructure, either as members of internal IT departments, or with IT service providers who remotely manage systems for customers. In addition to handling installation, monitoring, maintenance, upgrades, and other tasks, one of the most important jobs of SAs is to diagnose and solve problems.

In IT services environments, the *problem management process* (defined, for example, in ITIL [4]) describes the steps through which computer problems are reported, diagnosed, and solved. A typical sequence is for a problem ticket to be opened by a call to the customer helpdesk, or by an alert generated by a monitoring system. This is followed by some basic diagnosis by first level support personnel based on, for example, well-documented procedures. Simple issues such as password resets or file restoration can often be handled here without progressing further.

If the problem needs further investigation, it is passed to second or third level personnel, who are typically SAs with more advanced skills and knowledge. They often start with vague or incomplete descriptions of problems (e.g., “application is running very slowly,” “mail isn’t working,” or “CPU threshold exceeded”) which require significant investigation before the cause and solution are found. In the context of server support, administrators often consult monitoring tools that provide some specific system indicators, and then log in to the server to collect additional detailed information using system utilities. In the course of day-to-day problem management, this process is often the most time consuming and expensive task for SAs – it

is difficult to automate, and requires field experience and expert knowledge.

Unlike the first level support personnel, there is hardly any well-documented procedure one can refer to during this advanced problem management process. SAs usually rely on their own knowledge and experience to diagnose the root cause of the problem. Because of the complexity of the problems, it is a significant challenge to create a useful documentation about this process which can be referred by others. Especially in the environment where supporting teams are globally distributed, how to create and share this knowledge is a big challenge.

In this paper we describe Problem Determination Advisor (PDA), a system management tool for servers that provides health indicators and automated problem diagnosis capabilities to assist SAs in solving problems. PDA is intended to be used by second or third level SAs who diagnose and address problems that cannot be handled by first level support (i.e., helpdesk). PDA uses a two-level approach that provides high-level health monitoring of key subsystems, and *scoped probing* that collects additional details in an on-demand, rule-based fashion. This approach has the advantage of performing detailed “drill-down” probing only when it is relevant to the problem at hand, hence avoiding the overhead of collecting such data all the time. Moreover, PDA’s probes and problem determination rules are not determined arbitrarily; they are crafted based on an extensive empirical study of real problems that occur in practice and expert rules drawn from system administrator best practices. The rules are also customizable to better serve a particular environment or purpose. Our specific contributions include:

- *Characterization of server support problems:* using real problem tickets from a diverse group of servers in a large enterprise, we study the relative frequency of various types of problems, and determine categories of commonly occurring problems based on their nature and frequency, and study the relative difficulty of resolving different types problems.
- *Problem determination rules for scoped probing:* based on examination of actual problem descriptions and their solutions, tools and scripts used by SAs in practice, and knowledge capture from SAs, we develop a set of rules that determine how probes should be dispatched to automatically collect the information necessary to assist the diagnosis of various types of problems.
- *Extensible PDA tool architecture:* we codify problem determination rules and best practices in a tool that can be expanded as new rules are developed or as specific needs arise in different IT environments.

The measure by which most system management tools, and problem determination tools in particular, are judged is the reduction they enable in the time or effort needed to solve problems. As such, we illustrate the usage and benefits of PDA with a number of UNIX problem scenarios drawn from problem tickets, discussions with SAs, and system documentation. In these scenarios, SAs must know which information to collect and how to collect it (manually) in order to solve the problem. We show that PDA is able to quickly collect key information through its problem determination rules to aid in finding the cause, or in some cases pin-pointing the problem precisely. This allows expert SAs to solve problems more efficiently, and less experienced SAs to benefit from the diagnostic best practices codified in the tool.

Our current library of problem determination rules and probes is geared toward system software problems, for example on commercial and open source UNIX platforms. However, the general approach of PDA is applicable to other problem areas, particularly applications and middleware (which are a significant fraction of all problems). As we discuss in Section Problem Characterization, the bulk of reported problems are related to a relatively small number of specific problem areas, and this holds across problems related to applications, platform, networking, etc. Hence, developing rules based on best practices for the most commonly encountered problems is quite feasible. However, we expect that this model is more beneficial for IT services environment where similar platform and configuration co-exist. For heterogeneous environment such as universities, the best practices may vary from each other.

In the next section, we present the results of a characterization study of problem tickets which we use to inform our choice of system probes and the design of

problem determination rules. Section PDA Design and Implementation follows with a description of the PDA tool design and implementation. In Section Problem determination experiences with PDA we evaluate PDA's efficacy in the context of several specific problem scenarios. Section Related work briefly discusses some of the related work. We conclude the paper in Section Summary with a discussion of our ongoing work in the implementation and evaluation of PDA.

Problem Characterization

We begin with a characterization of real-life problems from a large, distributed commercial IT environment. The problems are drawn from an analysis of about 3.5 million problem tickets managed through a problem tracking system over a nine-month period. These tickets contain rich amount of information including a problem description, indication of the affected system, and metadata such as timestamps, severity, and identity of the SAs handling the tickets. In our analysis, we derive a number of statistical characteristics that allow us to better understand the different categories and characteristics of common problems. Specifically, we focus on:

- which applications contribute to the majority of the problems, and in particular whether a few applications are responsible for most of the observed problems
- what are the most common causes of application problems
- what portion of problems arise due to application vs. operating system-level issues
- how much time is spent in resolving different types of problems

Our objective in this section is to develop insights from the analysis of problem tickets in a real enterprise IT environment, and later use these insights to develop tooling for improving the efficiency of problem diagnosis and resolution.

Fields in Problem Tickets

We examine a number of attributes of each ticket including both structured attributes with well-defined values, and unstructured attributes which are mostly free-text. Structured attributes contain information such as a ticket's open and close time, incident occurrence date, SA user ID, and some enumerated problem characteristics such as problem type, product name, etc. Problem description and solution description are free-text. In this study, we are particularly interested in the following data fields:

- *product name:* There are about 600 products appearing in the tickets. Most of them are application names, while operating system or platform-related problems are categorized by general terms such as AIX, HP-UX, Linux, Windows.
- *product component:* Each product is further broken down into various predefined components.

They give finer-grained information about the problems. For example, components within the Windows product include bootup, explorer, system errors, password.

- *product module*: This is the finest-grain information available in the problem database – the module identifies the system sub-component that is having the problem. For example, in Windows, the bootup component is divided into different modules such as safe mode, unable to boot, inaccessible boot, and explorer contains navigation, move/copy Files, search.
- *ticket type*: When a ticket is first opened, it is categorized into a type, for example: error, performance, information request, load request and others. When studying common problems, we focus on those tickets with type of error and performance because they usually require further diagnosis to identify the root cause.
- *cause code*: This field has 34 values to describe various problem causes. It is particularly useful when other fields such as component and module are not specified. For example, all of the tickets related to the Linux product name have *software* as the component field and *OS* for module.

Problem Characteristics Overview

To understand the problem tickets in the system, we start from statistical characteristics of the tickets based on the above-mentioned fields. We make the following observations:

- **Most of the problem tickets arise from a few products.**

We examined all tickets according to the product name field. Among the 600 products, we observe that 50 products, which are less than 10% of the total products, account for 90% of all tickets. Figure 1 shows a cumulative frequency graph of the number of products and the percentage of the tickets with that product. This observation is similar to the failure characteristics observed on Windows XP machines [5]. Among the top 50 products with the most problem tickets, the number one product is an enterprise email system, followed by a virtual private network (VPN) application, and then a popular operating system. Figure 2 shows more detail about the number of tickets and their distribution, and Table 1 shows the top 10 products and the number of tickets from each.

- **Within each product, most of the problems come from a few modules.**

Next, we analyze the top products which have the most tickets by categorizing them according to product components and modules. Table 3 lists the 18 modules which comprise 70% of the tickets reported for the mail system, while the total number of modules defined for this product is

162. For the VPN application, we see an equally skewed distribution: 70% of the tickets are from six modules, out of a total of 70 modules. Table 2 lists these top six modules. Details of product components are not listed here because they reveal less information than the modules.

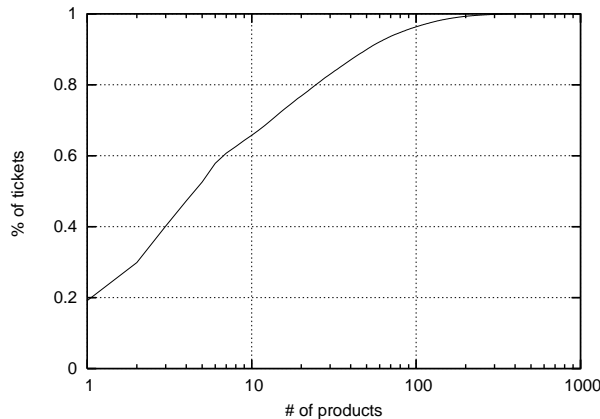


Figure 1: Cumulative frequency of the tickets contributed by the set of products.

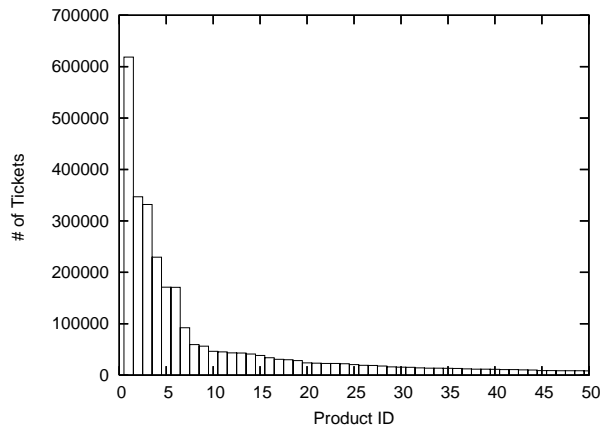


Figure 2: Number of tickets from the top 10% products with the most problem tickets.

These two observations together suggest that, by focusing on problem determination for a relatively small number of products and corresponding modules, we are able to cover a large portion of problem tickets. However, it is also important to note that each module may in fact exhibit many problem symptoms and possibly many root causes. This implies that a practical tool should be highly customizable in order to address symptoms that might be unique to a particular environment.

Operating System Problems

In terms of quantity of problem tickets, operating system (OS) problems are not as prominent (with the exception of Windows-related tickets, which consist of about 11% of the all tickets). However, system software or OS tickets are often the most diverse and require significant effort to diagnose. Figure 3 illustrates

the distribution of time spent in resolving problem tickets for top applications and systems/OSes. Our analysis indicates that nearly an order of magnitude more time is spent in the resolution of problems arising from UNIX system issues, than on other types of problems. VPN and mail applications, both comprising a large number of application-related problem tickets, need considerably less time to resolve compared to UNIX system related problems.

Product name	Cumulative # of tickets	% of tickets
Mail app	618575	18%
VPN app	346812	28%
Windows OS	331978	38%
Mail app (prev version)	229415	45%
PC hardware	171078	53%
Network connectivity	170937	58%
Software installer	92358	61%
Mainframe local app	59314	63%
Telephone	56396	64%
Desktop security audit tool	46470	66%

Table 1: Top 10 products in the ticket database.

Product name	Cumulative # of tickets	% of tickets
RESET	114477	38%
LOOPING	23620	45%
INTRANET APP	22045	53%
INFO	20537	60%
INTER/INTRA NET	14220	64%
CLIENT	13409	69%
UNABLETOCONNECT	13390	73%

Table 2: Product modules and the number of tickets from these modules for the VPN application.

We also examine the top problem types for OS platforms. Unfortunately, there are no OS components and modules defined except for Windows (perhaps due to the complexity of OS problems). Instead, we use the cause code field in analyzing OS tickets. We combine similar cause code values to arrive at a set of broader problem categories including: application, configuration, hardware, request/query (e.g., password reset or howto questions), duplicate (i.e., multiple reported problem), storage, network, human error, unsupported (i.e., out of scope for the support team). Note that “application” in the OS tickets differs from application as standalone product – these are mostly system processes or services shipped with the OS such as Sendmail, NFS, Samba etc.

Common OS Problems

From the problem categories in Figure 4, we further examine ticket details in a few categories to

identify commonly occurring problems that occur in each category. In particular, we focus on problem categories related to systems software and application-related issues on the UNIX platform, including: application, configuration, storage, and network. We use a combination of ticket clustering based on structured attributes, and manual inspection of problem and solution description text, to extract a set of typical problems. We describe a few of these sample problems below.¹ Recall that the problems are grouped by cause code values that are indications of the identified cause; this may be a different category than the original problem description would initially indicate.

Product name	Cumulative # of tickets	% of tickets
OPENING DB	54745	12%
RESET	34816	20%
FEATURES	32775	27%
INSTALL/SETUP	25365	32%
OSM/FILESIZE	23458	37%
OPTIONS	17788	41%
SETTINGS	17453	45%
SEND/RECEIVE	17013	49%
HELP	13785	52%
TEMPLATE	13049	55%
CHANGE	12229	57%
SETUP	10631	60%
CREATE ID	9348	62%
SENDING	8786	64%
OPTIONS/PREFS	8385	65%
NOTRESPONDING	7086	67%
GENINFO	7034	69%
AUTHORIZATION	6452	70%

Table 3: Product modules and the number of tickets from these modules for the mail application.

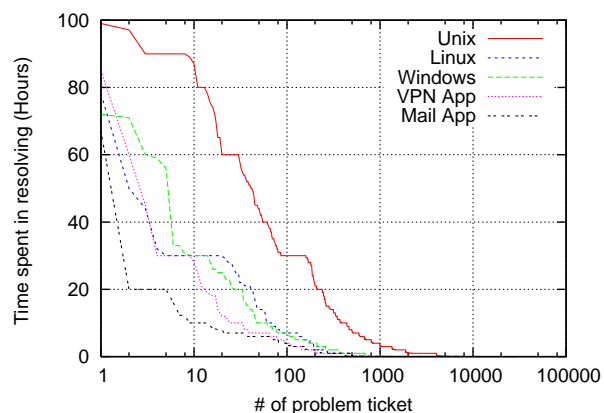


Figure 3: Time spent in solving tickets in top two products and in three OSes.

¹Specific hostnames, directory names etc. have been anonymized in these samples.

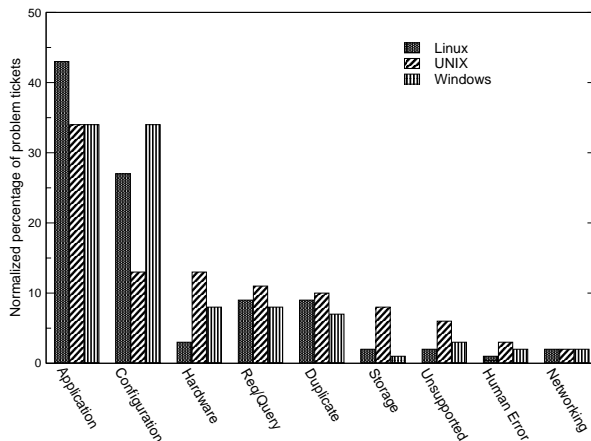


Figure 4: Categorization of problem tickets for Linux, UNIX and Windows servers. We examined 4,598, 42,998, and 331,978 tickets for each platform, respectively.

Configuration Problem Samples

- *Multiple email messages from dissimilar domains appear to be blocked.* The reason was due to a change in spam filtering mechanism/update.
- *SSH failing after OS version upgrade.* The solution of this problem was to change permissions on /dev/random and /dev/urandom to 644
- *The following NFS filesystems did not mount after miwsvr1 rebooted today:*

```
efkxeastapps.abc.xyz.com:/u0
efkxeastapps.abc:/u1
efkxeastapps.abc.xyz.com:/u2
efkxeastapps.abc.xyz.com:/u3
efkxeastapps.abc.xyz.com:/u4
```

The reason of this problem was because the filesystem has been updated using another type. Solution was to mount the filesystem using that type name.

Application Problem Samples

- *the databases will not start because of some TCP/IP problem.*
- *03:38 OPS received the following red alert on UNIX icon for abcdef03:*

```
Host:named.my.ibm.com
Msg:The percentage of available swap
space is low (20.12939453125 percent).
```

The solution was to stop unwanted processes.

- *SRM data is not getting retrieved from STI-BACKUP due to the refusal of the SFTP connection. Symptoms suggest SSH is not running.* The problem was solved by changing /usr/bin link to ssh and /etc/inetd.conf path then restarting ssh.

Storage

- *Base_OS_Monitors critical 98.006800 Percent space used (/var) This Critical Sentry2_0_disk*

usedpct event was received by a monitoring server at 3/6/2006 6:20 EST.

Typical solutions to this space usage problem include removing old files, large files, expanding file system space, and compressing the old files, etc.

- *System backup failed for sx0000e0 on 18082006. More details could be found in the logfile /var/adm/mksysb.out.*
Reason for this problem was because some temporary files were not found.

A summary of examples of top problems that we found among problem tickets is shown in Table 4 in the first column.

Implications From Problem Ticket Analysis

Our observations in characterizing problems in a large IT environment has a number of implications that guide the design of PDA. Recall that the first key finding was that a few products or applications are responsible for the majority of problem tickets opened. The second observation was that the cause of these problems can be attributed to a relatively small set of functional components of these products. These results together imply that problem determination tooling that addresses a finite and fairly small set of important problem types can in fact cover a significant portion of problem tickets that are observed in practice.

We also observed that, in terms of time spent on resolution, UNIX system related problems are relatively difficult to resolve. Therefore, this is an important problem area to consider. Improving diagnosis efficiency for UNIX-related problem can potentially provide a significant value in terms of reduced time and effort.

These observations motivate our implementation of PDA, which addresses a number of key categories of system software or OS-related problems. While our intention is to broaden the applicability of PDA to other problem types (e.g., applications), our initial focus on OS problems on UNIX platforms is justified based on the analysis presented above.

PDA Design and Implementation

In this section, we first describe the overall design, architecture, and implementation of Problem Determination Advisor. We also describe how knowledge gathered from problem tickets is incorporated in PDA's automated problem diagnosis capabilities. Details of problem determination rules and system probes, as well as some realistic examples, are also illustrated below.

Design Overview

From the previous section, we have observed that a large percentage of problem tickets are related to a small number of products, and within each, most problems have only a few primary causes. We use a two-level approach that provides high-level health monitoring of key subsystems, and scoped probing

that collects additional system details. In Table 4, the second column lists some of the high-level monitors that are used in practice to identify the occurrence of common problems, while the third column shows examples of diagnostic probes that can help identify the cause of common problems in the corresponding category. The list of problems reported here represents a sample of the problem scenarios we have examined, and is by no means complete. We are continuing to expand the list of common problems and corresponding problem determination probes as we examine additional tickets and continue the knowledge capture of SA best practices.

If the full complement of monitors and probes are always active (e.g., executing periodically), they would likely impose a noticeable overhead on production systems. Hence, the two-level probing approach uses (i) periodic, low-overhead monitoring to provide a high-level health view of key subsystems, and (ii) detailed diagnostic probes when a problem is detected. This is similar to the way SAs tackle problems – the difference being that PDA tries to collect the relevant problem details automatically.

The knowledge of which diagnostic probes should be run when a problem is detected is encoded in problem determination rules. These rules are represented in a decision tree structure in which the traversed path through the tree dictates the series of diagnostic probes that are executed. At each node, the output of one or more diagnostic probes is evaluated against specified conditions to decide how to proceed in the traversal. In our implementation, diagnostic probes generally use available utilities on the platform directly to retrieve the needed information.

Problem Determination Rules

Figure 5 shows a sample rule tree which can diagnose problems related to the network connectivity of a managed server. The corresponding health monitor considers the system’s network connection to be available if it is able to reach (e.g., ping, or retrieve a Web page)

several specified hosts outside of its subnet. These could be other servers it depends on, or well-known servers on the Internet, for example. If a disconnection is detected by the health monitor, scoped probing using diagnostic probes will be invoked to gather information to help determine the root cause of the problem. According to the rule tree, the first diagnostic probe should check the network stack by pinging the loop-back address. If no problem is found, the next node in the rule tree will dispatch another diagnostic probe to check that a default gateway is defined in the local routing table, and that it is reachable. If it is unreachable, the problem might be with the local subnet or network interface card. Otherwise, potential DNS-related problems are checked, for example verifying that /etc/resolv.conf exists, and that it contains DNS server entries (that are reachable). Clearly some diagnostic probes have dependencies (e.g., check /etc/resolv.conf exists before checking that a DNS server is reachable) and have to be executed in a certain order. In the absence of dependencies, probes could be ordered differently, perhaps tailored to the likelihood of certain types of failures in a given environment.

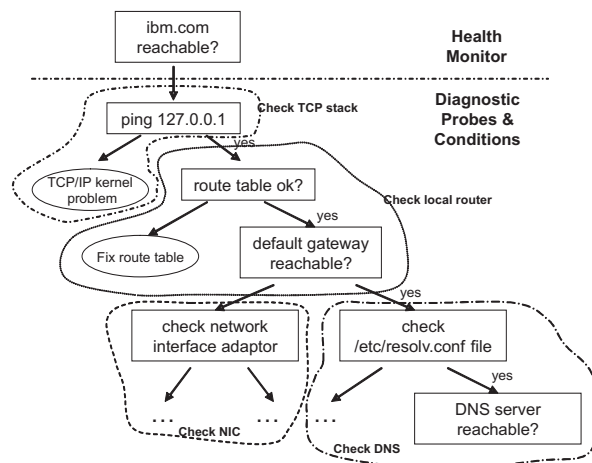


Figure 5: Sample rule for network-related problems.

Category	Typical Problems	Health Monitors	Diagnostic Probes
Configuration	OS/App upgrades, changes to various configuration files, firewall, spam filtering	Track changes to upgrades, configuration files	File privilege, active users, file diff with backup
Application	App errors due resource exhaustion (incl. CPU, memory, filesystem), app prerequisites, path/setup, etc.	Check resource usage, error log, process status etc.	Processes having most of the CPU, mem, IO, etc.
Network	Connectivity, performance	Check DNS, firewall, routing table, NIC	Traceroute, TCP dump, network options, NIC
Storage	Capacity, data corruption, mount problem, performance, disk swap, etc.	Check available space, I/O rate, error logs	Mount options IO history, big files

Table 4: Examples of common problem symptoms and corresponding monitors and probes used to identify these problems.

Our problem determination rules are primarily derived from inspection of problem tickets and by capturing best practices from SAs (i.e., through discussions, reviewing their custom scripts and procedures, etc.). In the case of problem tickets we extract rules by examining the steps through which a problem was diagnosed and resolved. When the detailed steps are available, creating a corresponding rule tree is fairly straightforward. Some of the tickets, however, do not have much detail beyond the original problem description and perhaps a few high-level actions taken by the SA. In such cases, we must manually infer the probes needed to collect the appropriate diagnosis data. In our rule extraction process, we use a combination of data mining tools to categorize problem tickets and identify distinguishing keywords, followed by varying degrees of manual inspection to better comprehend the problem and solution description text. The data mining tools are not described in detail here. Our experience with problem tickets so far leads us to believe that some amount of manual inspection is necessary to derive corresponding rules, however we continue to investigate automated techniques to assist the rule derivation process.

System Architecture

We have implemented a fully functional prototype of PDA, including the probing and data collection mechanisms, rule execution, and a Web-based interface. Figure 6 shows the overall architecture of PDA, which contains three major components: probe daemon, PDA

server, and the user interface. Our probe daemon is implemented in C for performance consideration and for easy deployment. PDA server is implemented in Java and our current user interface backend uses IBM WebSphere Portal Server. We use MySQL as our database storage.

Web User Interface

The Web UI allows SAs to perform various tasks from a single interface accessible from any workstation. It gives SAs an at-a-glance health overview of all of the servers being managed by clearly highlighting systems and components that have problems or are predicted to have problems in the near future. Whether or not an indicator implies a problem is determined by the corresponding rule, as discussed further below. In addition to showing the current health view of managed servers, we also allow SAs to look at the status of the servers at earlier points in time. This feature is useful when a reported problem is not currently evident on the system, but may be apparent when viewing system vitals collected earlier. It also is crucial for observing trends in certain metrics. Based on our discussions with SAs supporting commercial accounts, this feature is particularly useful to them in gaining a better understanding of the behavior of the managed systems.

Besides monitoring, the Web UI allows SAs to perform some simple administrative tasks such as adding another server to be monitored, updating a user's access privilege (as a root SA), adding or removing

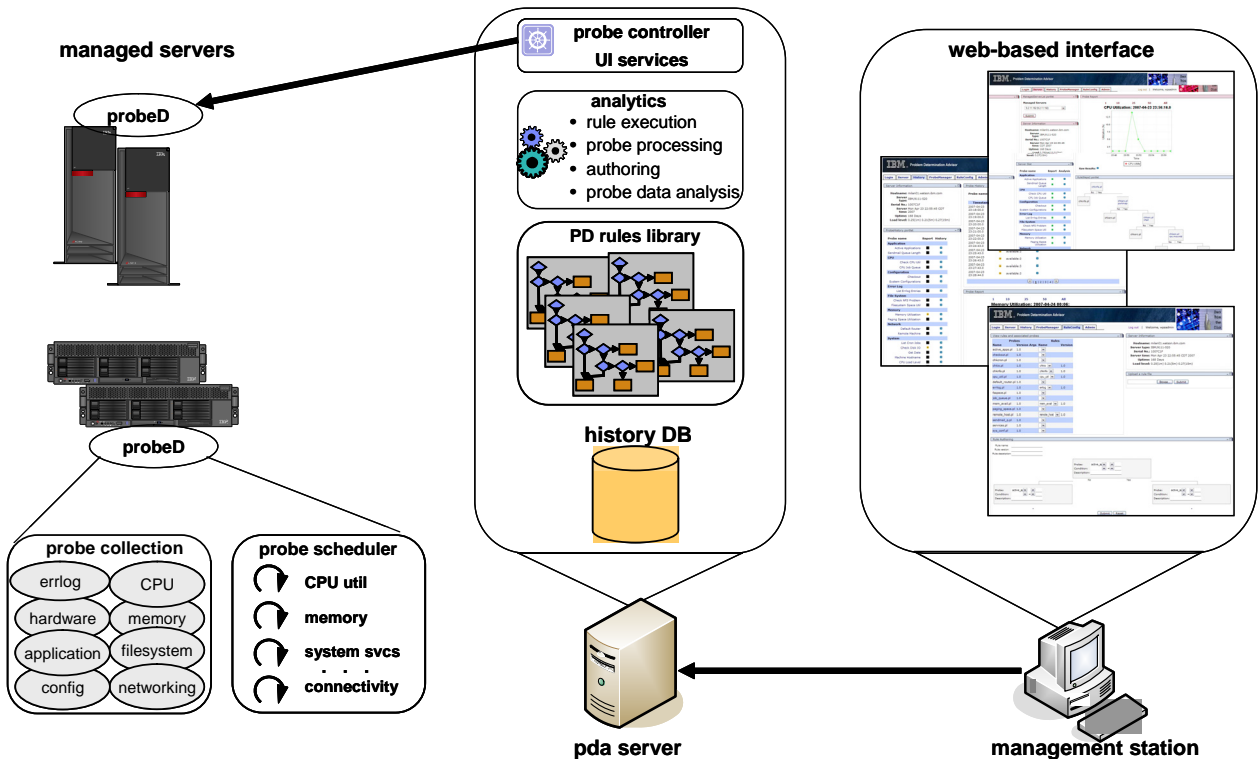


Figure 6: PDA architecture.

probes on a managed server, etc. The Web UI also provides an interface for SAs to author probes and construct rules from new and existing probes. Rules and probes constructed using this interface are stored in XML to facilitate sharing with other SAs or reusing them across multiple platforms. This feature is discussed in more detail in Rule Sharing Section.

We are also in the process of implementing a Web based secure login mechanism so that SAs can quickly switch from a shell from one machine to another using pre-defined credentials. This allows SAs to better visualize monitored data using the graphical interface, while still having access to a low-level command line interface from a single tool.

Probe Daemon

The probe daemon is a small program that is runs on managed servers whose primary functions are to schedule the execution of probes according to the frequency set by the SA, and to interface with the PDA server. The PDA server sends command to start a new probe, stop an existing probe, change the periodicity of a probe, etc. When starting a new probe, the probe daemon can download the probe from a central probe repository if the probe does not exist or is not up-to-date locally. After probes have finished executing, the probe daemon is also responsible to send probe results back to the PDA server.

PDA Server and Rule Library

Most of the information exchange and processing is handled by the PDA server. Periodically it receives probe results from the probe daemon, stores the results in a history database, and triggers rule execution if there is a corresponding rule for a particular probe. The history database stores collected data and also serves as a repository for important configuration files that are tracked by PDA.

The rule execution engine is the most important part of the PDA server. It parses rules, each defined in a separate XML file located in the database, and converts them to an in-memory representation for evaluation. There are two ways to evaluate a rule. Typically, a rule is triggered by a periodic probe which is defined to be the root node of the rule tree. The trigger can be a threshold violation, change in a key configuration file, or other detected problem. As a second method, an SA can execute a rule to initiate it manually, to proactively collect information related to specific subsystem.

In both cases, as the rule tree is traversed, a command is sent to the probe daemon to execute the required probe and return the result. The result is used to make a decision to continue collecting more information or stop the rule execution. Some rules also use historical data to decide what should be collected next, or what should be displayed to the SA.

Since PDA supports management of multiple groups of servers (e.g., for different customers), the PDA server also keeps track of which servers are

managed by which SAs. This also implies that each server or group can have different active rules and probes; PDA supports this notion of *multi-tenancy* in the rules library and rule execution engine.

Rules and Probes

A probe is usually implemented as a script (e.g., Perl, shell, etc.) that either executes native commands available in the system or interfaces with other monitoring tools deployed in the environment. The probe parses and aggregates the output of the commands, and returns the results as an XML document. In order to make it easy to add new probes to PDA, the schema is a simple and generic key-value pair representation. When interfacing with other monitoring tools to collect data, we write adapters to convert their output to the XML format for the PDA server.

Figure 7 is an example output from a probe that monitors Ethernet interfaces.

Rules are triggered automatically by a monitoring probe which appears as the first node of the rule tree. For example, Figure 8 shows a sample rule *chk_interface*. It will be triggered by a probe called *chk_eth*. In this rule, the first step tests if the number of collisions is beyond a certain threshold. If the threshold is exceeded, the next probe, *chk_switch*, is executed to collect some information about the network switch, for example related to the firmware version. This type of scoped probing minimizes monitoring overhead and expedites the problem determination process. In the case where the probe in the first node does not present, or the problem is reported by other channels, such as problem ticket, a rule can be executed manually by the SAs.

```
<results probename="chk_eth">
  <result>
    <key> INTERFACE </key>
    <value> eth1 </value>
  </result>
  <result>
    <key> ERRORS </key>
    <value> 0 </value>
  </result>
  <result>
    <key> DROPPED </key>
    <value> 0 </value>
  </result>
  <result>
    <key> COLLISIONS </key>
    <value> 50234 </value>
  </result>
</results>
```

Figure 7: A sample probe output.

Probe and Rule Authoring

Given the heterogeneity of enterprise systems and applications, it is unrealistic to expect a single library of rules and probes to work in all IT environments. For this reason, PDA is designed to be

extensible to allow authoring of probes and rules, either from scratch or, more commonly, based on existing content. We provide templates for writing new diagnostic probes, and a way to logically group rules and their associated probes (e.g., based on a particular target application). Being able to quickly construct a rule for an observed problem from a set of existing probes can be very helpful to save SAs precious time.

We have implemented a Web-based probe and rule authoring interface. We validate the input of restricted fields and perform some simple checks (e.g., for duplicated names in the repository). Validation of the probe code is similarly simple, comprising checks that the probe runs successfully and implements the necessary rule output formatting. Newly created rules require slightly more involved validation. For example, we validate that each node in the rule tree has the corresponding probe(s) available, and that the conditions being checked are supported by the probe.

Once a new probe is authored, its data fields are created and stored directly in the corresponding tables in the database, and the PDA server is notified of the new probe name. If the probe is periodic and needs to be activated for monitoring, the probe daemon is contacted by the PDA server automatically to schedule the probe. If the probe is for diagnostics (i.e., a “one-time” probe vs. periodic), it will be downloaded to the managed server when it is invoked by a rule. Rules are stored similarly, along with the XML representation of the tree structure.

Since our probes are mostly scripts which will be running on managed servers, they pose a potential security threat to the system if probes are malicious. Currently we rely on user authentication and out-of-band change approval for new probe and rule authoring. It may also be feasible to use compilation techniques to perform some checks on the semantics of the scripts, for example to see if a probe is writing to a

restricted part of the filesystem. We are investigating this in our ongoing work on PDA.

Rule Sharing

In our discussions with SAs, we found that sharing knowledge and experience between them is a considerable challenge. One potentially significant benefit of rule and probe authoring in PDA is the opportunity to share them with other SAs managing the same environment, or even those working in very different environments.

Guaranteeing that rules and probes authored by one SA are applicable to problem resolution on other systems poses a number of difficulties. The primary one is the wide variety of platforms, operating systems, and software. Most rules are largely platform-independent as the information can be extracted on most OSes. However, the probes that actually collect the information can be quite different on various platforms. Even on machines with the same OS, different patch levels or software configurations can very easily break probes. To make sharing of rules and probes more seamless to SAs, we annotate them with dependency information that indicates the platform and version on which they have been deployed or tested.

Initially, we expect to deploy our tools in a fairly homogeneous environment, e.g., with mostly UNIX machines. We expect most dependency issues in such an environment to be solved relatively easily, for example by using a different binary/utility to obtain the same information. This technique can be carried over to managing other flavors of UNIX or Linux. As more users contribute rule and probe content over time, they will likely cover a more comprehensive set of platforms and provide a valuable way to accumulate and codify system management knowledge.

PDA Usage Model

The design of PDA is largely motivated by our experience in IT service provider environments, in

```
<rule rulename="chk_interface">
  <node probename="chk_eth" id="0">
    <condition> COLLISIONS > 500 </condition>
    <true-branch> id="1" </true-branch>
  </node>
  <node probename="chk_switch" id="1">
    <condition> MANUFACTURER == LINKSYS &&
      MODEL == ETHERFAST &&
      FIRMWARE_VERSION <= 2.3.1
    </condition>
    <true-branch>
      alert("upgrade firmware")
    </true-branch>
    <false-branch> id="2" </false-branch>
  </node>
  <node ...>
    ...
</rule>
```

Figure 8: A sample rule file.

which globally distributed support teams manage the infrastructure belonging to a large enterprises. In these environments, creating, communicating, and adhering to best practices for systems architecture and management is a significant challenge. Global support teams often consist of administrators with greatly varying amounts of experience and knowledge – providing the ability to capture and operationalize problem determination procedures for the whole team is very valuable. In addition to improving efficiency through automated collection of relevant information, it also allows SAs to follow similar procedures which could be designed by the most experienced team members. At the same time, PDA allows customization of problem determination rules to account for different customer environments or priorities. A PDA installation could include a standard set of problem determinations rules and associated probes that handle common or general problems (e.g., networking problems, excessive resource consumption, etc.). These could be supplemented with new content that is available from a central repository, or through local modifications of existing content.

This usage model is particularly applicable for IT service providers who manage many customer infrastructures in a number of industries, since it is likely that there is a lot of similarity at the system software level. Furthermore, when the service provider has performed some degree of transformation in the customer environment, for example to move to preferred platforms and tools, the possibility for sharing and reuse increases. In IT environments belonging to universities and industry research labs, we observe more heterogeneity in OS platforms, applications, and usage, which makes it more difficult to develop problem determination best practices that are widely applicable. Nevertheless, the automation and extensibility features of PDA are still useful in these situations.

Problem Determination Experiences With PDA

This section describes some of our experiences in constructing rules and probes to diagnose practical problems with PDA. The rules we built based on problem tickets and best practices from interviewing SAs are not comprehensive, but they address commonly occurring problems in realistic settings and can be expanded and enhanced by communities of users or administrators.

Experience with NFS Problems

NFS allows files to be accessed across a network with high performance, and its relatively easy configuration process has made it very popular in large and small computing environments alike. However, when problems occur, finding the root cause can take a significant amount of time due to NFS's many dependencies. Most of the NFS-related problem tickets we observed are straight-forward to solve, but some have symptoms that are difficult to connect with their final solution. This often results in tickets being forwarded

multiple times to different support groups, sometimes incorrectly, before they are finally resolved. We found that most of these problems can be diagnosed with a few simple systematic steps.

```

$raw = `lssrc -a`;
$lines = split(" ", $raw);
# Omitted code for error-checking
#   executing lssrc
# Parses lssrc output: 3 possible
#   output formats
# 1. Subsystem Status
# 2. Subsystem Group Status
# 3. Subsystem Group PID Status
foreach $line (@lines)
    $line =~ s/^(s+|s+$)/g;
    @lineElem = split(/s+/, $line);
    $numLineElem = scalar(@lineElem);
    $match = $lineElem[0] eq $ARGV[0];
    if ($numLineElem == 4)
        if ($match &&
            $lineElem[3] eq "active")
            $foundActive = 1;
        elsif ($numLineElem == 3)
            if ($match &&
                $lineElem[2] eq "inoperative")
                $foundButNotActive = 1;
        elsif ($numLineElem == 2)
            if ($match &&
                $lineElem[1] eq "inoperative")
                $foundButNotActive = 1;
# Format output and dump to stdout
if (defined($foundActive))
    &PDAFormatOutput(...);
elseif (defined($foundButNotActive))
    &PDAFormatOutput(...);
else
    &PDAFormatOutput(...);

```

Figure 9: A simple probe that checks if a system service is currently running.

The rule to determine NFS-related problems is shown in Figure 10 as a tree. The rule tree is traversed by information gathered from dispatching a series of probes. Some rules are intuitive – check for liveness of all NFS service daemons, e.g., `nfsd`, `mountd`, `statd`, and `lockd`, and check if these services are properly registered with the portmapper. In Figure 9, we show an example probe that checks if a system service has been started and is currently running. As each probe does something very specific, it can be quickly and easily implemented and maintain. We used Perl to implement this probe, but probes can be written in any language as long as their output format matches the pre-defined format. There are also other more esoteric rules – `statd` should always start before `mountd`, or the `exname` option can only be used if the `nfsroot` option is specified in `/etc/exports`. However, we have seen that a majority of problem tickets can be addressed by the simpler checks, e.g., looking for a missing `/etc/exports` file, non-existent mount points, etc. Some probes are useful to exercise periodically to help SAs maintain a

healthy NFS service (e.g., check for hung daemons or abnormal RPC activities). Some can be run when changes are made to configuration files to check for potential problems caused by a recent change. Some can even be triggered manually in response to client-reported problems.

The benefits of PDA's automated problem determination capability are best illustrated by considering a specific problem scenario. Consider the problem of a misconfigured system that starts `nfsd` before `portmapper`. The misconfigured system will not allow users to access remote NFS partitions and the resultant problem ticket has a correspondingly vague problem description. In the absence of PDA, the SA must manually narrow down the problem cause, starting for example, by logging into the affected systems, checking network connectivity, verifying firewall rules, etc. to make sure the NFS problem is not a side effect of another problem in the system. Having done that, the SA may then check `/etc/exports` for possible permission problems, see if all the defined mount points exist and run the `ps` command to confirm that all NFS-related services are running. He or she might need to run more diagnostic tools before finally discovering that `nfsd` was not correctly registered with the `portmapper`. Using PDA however, this misconfiguration problem can be discovered quickly by examining the results of the automated rule execution.

Experience With Storage Problems

We were surprised that a large percentage (90%) of the storage-related problem tickets we observed were related to simple capacity issues, e.g., disk partitions used for storing temporary files or logs being close to or completely full, thus hampering normal operations in the system. Given the large volume of such tickets, early detection and resolution of storage capacity problems can potentially eliminate a large number of tickets. In many environments, management tools monitor filesystems and raise an alert if the

utilization crosses a specified threshold. This approach has the disadvantage of potential false alarms if for example, a filesystem is normally highly utilized (e.g., an application scratch area). In PDA, we use a low-overhead profiler that periodically samples disk usage for each partition being monitored and have the profiler raise an alert if a partition's usage (within a time interval N) has an upward trend that is projected to be completely filled within the next 24 hours. A 24-hour period is chosen to allow an SA enough time to confirm a problem and take action. An example illustrating how the profiler is able to find storage capacity problems is shown in Figure 11. In this example, simple linear regression is used for trending, which will detect that in interval 3 this partition will be full within a day and raise an alert. More complex regression methods can be used to further suppress false alarms.

As an example similar to what we described in Section 2.4, consider the actual problem ticket with this description: *Ops received the following alert on host: \$< \$hostname\$> \$ "Percent space used (/var/log) greater than 95% - currently 100%."*. This problem ticket was opened by the operations team after an alert was raised by a monitoring tool. Even though the problem may be easy to solve, by the time the SA receives this ticket, the system may already be having problems for an extended time, disrupting its normal operations. Using the profiler, such problems can be accurately predicted and notifications can be provided to SAs earlier.

Experience with Application Problems

Application problems are the most frequently encountered category of problems (as shown in Figure 4), and also the most varied due to the myriad applications running in enterprise environments. Some application problems can be detected with periodic probes similar to the way we diagnose NFS-related problems. These can check, for example, for the liveness of application processes, whether specific application ports

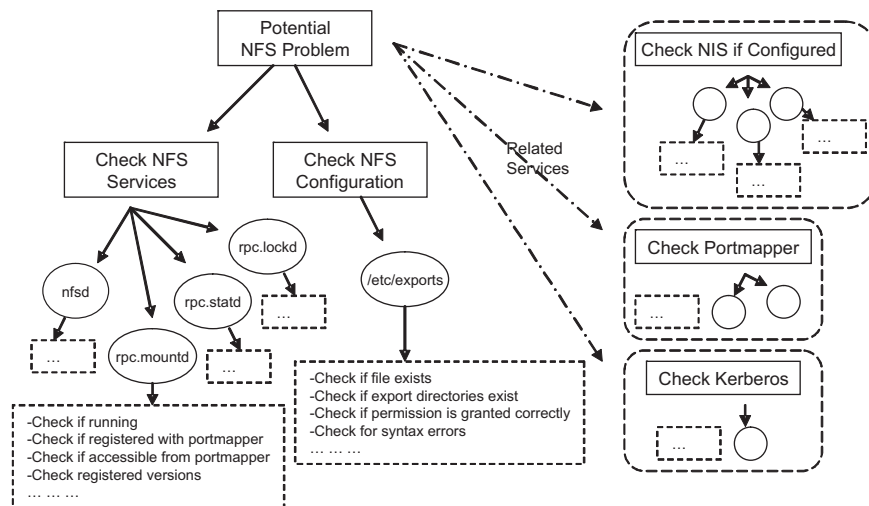


Figure 10: NFS problem determination rule tree.

are open, or for the presence of certain application files. The extensibility of PDA can be leveraged to handle other types of application problems, such as newly discovered security vulnerabilities. A community of users can devise and distribute new probes and associated rules to detect new vulnerabilities, which can shorten the system exposure time and save SAs time in evaluating whether their systems are affected.

One application for which we observed a significant number of problem tickets was the widely deployed mail server application, Sendmail. Traditionally, Sendmail configuration has been considered very complex and we expected the problems to be related primarily to setup issues or misconfiguration. However, again, most of the problems were actually caused by relatively simple issues – a congested mail queue, a dead Sendmail process, a newly discovered security vulnerability, etc. An example problem ticket related to Sendmail had the following description: *Ops received the following alerts for <hostname> advising: "Total mail queue is greater than 15000 – currently 56345. Sendmail server <hostname> is not responding to SMTP connection on port 25."* Using a periodic liveness probe and a profiler (similar to the one for filesystems described above), the SA can be notified of a pending problem well before a critical level is reached. In the current implementation, PDA incorporates rules to check some Sendmail-related parameters, including process liveness and mail queue length.

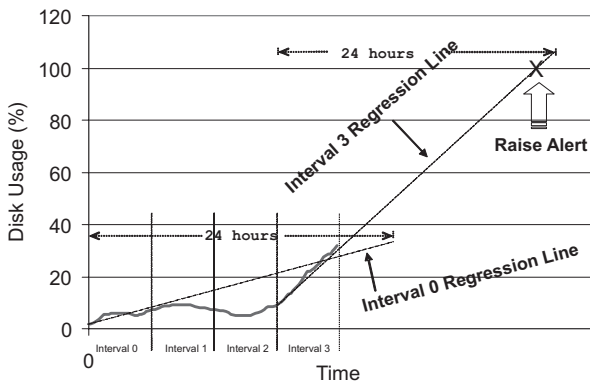


Figure 11: Using profiler to find storage capacity problems. The time axis is not drawn to scale.

Related Work

Though in certain cases, problems can be solved by fixing the underlying systems. However, a large quantity of problems addressed by SAs are not system bugs, instead, they are configuration problems, compatibility issues, usage mistakes, etc. Much of the prior work on system problem determination has centered on system monitoring. While monitoring is clearly important for PDA, it is not a primary focus of our work. We designed PDA to be able to use data collected from any monitoring tool. Our analysis of the ticket data helps us choosing which part of the system

should be monitored, but not how to implement monitoring probes. Our focus is instead on creating rules that help in diagnosing a problem once the monitoring system detects an issue.

Recently, researchers have been building problem monitoring and analysis tools using system events and system logs. Redstone, et al. [8] propose a vision of an automated problem diagnosis system by observing symptoms and matching them against problem database. Works such as PeerPressure [11], Strider [12], and others [7] [6] address misconfiguration problems in Windows systems by building and identifying signatures of normal and abnormal Windows Registry entries. Magpie [2], FDR [10], and others [13, 3] improve system management by using fine-grained system event-tracing mechanisms and analysis. We have similar goals to these works but our approach is to bring more expert knowledge in diagnosing problems and allow for a dynamically changing set of collected events.

While most problem determination systems have a fixed set of events to collect, few use online measurements as a basis for taking further actions. Rish, et al. [9] propose an approach called *active probing* for network problem determination, and Banga [1] describes a similar system on built for Network Appliance storage devices. Our system bears some resemblance to these approaches but is more generalized and has tighter link with observed problems, for example through IT problem tracking systems. In making decisions about what additional information needs to be probed, we use expert knowledge and indicators collected from real tickets as opposed to using probabilistic inference.

Our rules are conceptually similar to procedures, however, they are fundamentally different. Our rule combines knowledge and execution environment. The rule execution has intelligence to identify what is the next step and to execute the right diagnostic probes. Moreover, it is important to catch the system status when the problem just occurs. In current practice, SAs often need to reproduce a problem but the environment may have changed. So performing diagnosis right after the problem occurs not only saves time, but also could be the only way to catch the root cause.

Discussion and Ongoing Work

In our design of PDA, we were able to use problem ticket information as a guideline for the design of problem determination rules and associated probes. However, deriving rules from tickets still involves manual effort. We are investigating approaches to make this process more automated, but the varying quality of the free-text descriptions will be a continuing challenge. We are also investigating better models for the structured data which can more precisely capture problem signatures.

We use structured fields such as ticket type and cause code to categorize problem tickets and pick the

common problems as described in Section Common OS problems. Qualitatively speaking, our rules are able to address these common problems, but a more quantitative notion of “problem coverage” is required. One approach we plan to pursue to this end is to extract distinguishing attributes of a small set of tickets in the categories that are covered by our rules. The attributes can include a combination of structured fields as well as keywords from the unstructured text. Using these distinguishing attributes, we can examine a much larger set of tickets and look for matching tickets automatically to calculate the fraction of tickets that we expect to be similarly covered by the problem determination rules.

Although our experience with PDA shows promise that it can reduce time or effort for diagnosing problems, a more comprehensive study is needed to gain a sense of how much time or effort can be saved. We are making PDA available to system administrators who support a variety of customer environments in order to collect additional experiences, and ideally some quantitative data on the savings. We also expect to further validate and expand our problem determination rules through usage by SAs. Since there is no effective way to document problem resolution experience, our rule and authoring mechanism are good candidate for such a purpose.

Summary

This paper describes the Problem Determination Advisor, a tool for automating the problem determination process. Based on a study of problem tickets from a large enterprise IT support organization, we identified commonly occurring server problems and developed a set of problem determination rules to aid in their diagnosis. We implement these rules in the PDA tool using a two-level approach in which high-level system health monitors trigger lower-level diagnostic probes to collect relevant details when a problem is detected. We demonstrated the effectiveness of PDA in problem diagnosis using a number of actual problem scenarios.

Acknowledgments

We are grateful to our anonymous reviewers for their thoughtful and valuable feedback. We also want to thank our shepherd, Chad Verbowski, for his collaboration in improving the paper.

Author Biographies

Hai Huang is a Research Staff Member at IBM T. J. Watson Research Center. He worked on numerous power-management projects in the past, and his current interest is in system and application management and how to make the process more autonomous. He received his Ph.D. degree in Computer Science and Engineering from the University of Michigan. He can be reached at haih@us.ibm.com .

Raymond B. Jennings, III is an advisory engineer at the IBM T. J. Watson Research Center, Yorktown Heights, New York. He works in the area of network system software and enterprise networking. He received his BS in Electrical Engineering from Western New England College and his MS in Computer Engineering from Manhattan College. He may be reached at raymondj@us.ibm.com .

Yaoping Ruan is a Research Staff Member at IBM T. J. Watson Research Center. His research interests include system management, performance analysis and optimization, and server applications. He received his Ph.D. degree in Computer Science from Princeton University. He can be reached at yaoping.ru-an@us.ibm.com .

Ramendra Sahoo belongs to Systems and Network Services Group at IBM T. J. Watson Research Center. His research interests include business process management, distributed and fault-tolerant computing, numerical and parallel algorithms and data mining. He earned his B.E. (honors) from the National Institute of Technology, Durgapur, M.S. and Ph.D. from Indian Institute of Technology, Chennai and State University of New York at Stony Brook respectively. He can be reached at rsahoo@us.ibm.com .

Sambit Sahu is a Research Staff Member at IBM T. J. Watson Research Center. His interests include network and system management, performance analysis, network measurement, and Internet services. He earned his Ph.D. in Computer Science from the University of Massachusetts. He may be reached at sambits@us.ibm.com .

Anees Shaikh manages the Systems and Network Services group at the IBM TJ Watson Research Center. His interests are in network and systems management, Internet services, and network measurement. He earned B.S. and M.S. degrees in Electrical Engineering from the University of Virginia, and a Ph.D. in Computer Science and Engineering from the University of Michigan. He may be reached at aashaikh@watson.ibm.com .

Bibliography

- [1] Banga, Gaurav, “Auto-Diagnosis of Field Problems in an Appliance Operating System,” *Usenix Annual Technical Conference*, 2000.
- [2] Barham, Paul, Austin Donnelly, Rebecca Isaacs and Richard Mortier, “Using magpie For Request Extraction and Workload Modelling,” *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec. 2004.
- [3] Cohen, Ira, et al., “Capturing, Indexing, Clustering, and Retrieving System History,” *Symposium on Operating Systems Principles*, 2005.
- [4] Rudd, Colin, “An Introductory Overview of ITIL,” *IT Service Management Forum*, April, 2004, <http://www.itsmfusa.org> .

- [5] Ganapathi, Archana, Viji Ganapathi, and David Patterson, "Windows XP Kernel Crash Analysis," *20th Large Installation System Administration Conference*, 2006.
- [6] Ganapathi, Archana, Yi-Min Wang, Ni Lao, and Ji-Rong Wen, "Why PCs Are Fragile and What We Can Do About It: A Study Of Windows Registry Problems," *International Conference on Dependable Systems and Networks*, 2004.
- [7] Lao, Ni, et al., "Combining High Level Symptom Descriptions and Low Level State Information For Configuration Fault Diagnosis," *19th Large Installation System Administration Conference (LISA '04)*, Atlanta, GA, Nov., 2004.
- [8] Redstone, Joshua, Michael M. Swift and Brian N. Bershad, "Using Computers to Diagnose Computer Problems," *9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, Lihue, HI, May 2004.
- [9] Rish, Irina, et al., "Real-Time Problem Determination in Distributed Systems Using Active Probing," *IEEE/IFIP Network Operations and Management Symposium*, pp. 133-146, Seoul, Korea, April 2004.
- [10] Verbowski, Chad, et al., "Flight Data Recorder: Monitoring Persistent-state interactions To Improve Systems Management," *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, Seattle, WA, Nov., 2006.
- [11] Wang, Helen, et al., "Automatic Misconfiguration Troubleshooting With Peerpressure," *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec., 2004.
- [12] Wang, Yi-Min, et al., "Strider: A Black-Box, State-Based Approach to Change and Configuration Management and Support," *17th Large Installation System Administration Conference*, 2003.
- [13] Yuan, Chun, et al., "Automated Known Problem Diagnosis with Event Traces," *EuroSys*, 2006.