EXPERT SYSTEM FOR UNIX SYSTEM RELIABILITY AND AVAILABILITY ENHANCEMENT

Catherine Q. Xu, Ph.D.
Senior Staff Engineer
Aeronautical Radio, Inc.
Annapolis, MD 21401

ABSTRACT

Highly reliable and available systems are critical to the airline industry. However, most off-the-shelf computer operating systems and hardware do not have built-in fault tolerant mechanisms, the UNIX workstation is one example. In this research effort, ARINC has developed a rule-based Expert System (ES) to monitor, command, and control a UNIX workstation system with hot-standby redundancy. The ES on each workstation acts as an on-line system administrator to diagnose, report, correct, and prevent certain types of hardware and software failures. If a primary station is approaching failure, the ES coordinates the switch-over to a hot-standby secondary workstation. The goal is to discover and solve certain fatal problems early enough to prevent complete system failure from occurring and therefore to enhance system reliability and availability. Test results show that the ES can diagnose all targeted faulty scenarios and take desired actions in a consistent manner regardless of the sequence of the faults. The ES can perform designated system administration tasks about ten times faster than an experienced human operator. Compared with a single workstation system, our hot-standby redundancy system downtime is predicted to be reduced by more than 50 percent by using the ES to command and control the system.

INTRODUCTION

Product reliability and availability are the two most important qualities that ARINC has been pursuing. Currently, a number of systems and products are developed on computer systems running under the UNIX Operating System (OS). Because most off-the-shelf UNIX workstations are general-purpose machines, no fault tolerant mechanisms are built into either the operating system or the hardware. Therefore, hardware or software failures are not automatically detected by the OS utilities. In other words, most UNIX systems do not have any built-in *intelligent* fault diagnostics, fault-correction, or failure-prevention capabilities.

This research project develops a monitor, command, and control ES acting as an on-line system administrator to detect, report, correct, and prevent certain types of hardware and software failures on a UNIX workstation system with hot-standby redundancy. The goal is to use the ES to discover and solve certain fatal problems early enough to prevent system failure from occurring and therefore enhance the system reliability and availability.

The reason we are investigating the potential of using the ES to monitor, command, and control the UNIX workstation is because we need software to perform some of the system administrator's jobs in a real time, automatic fashion. Usually system administrators use *rules of thumb* logic gained through experience to solve problems. ES are the most effective approach to capture and use *rules of thumb* logic to solve problems.

SYSTEM OVERVIEW

As shown in Figure 1, a redundant workstation system consists of two processors—A and B—both powered on, where processor A is the original primary machine and processor B is the secondary. The two processors communicate via StarLan.

One ES resides on each machine, and each ES accomplishes two tasks: (1) process self-checking and hardware management and (2) hot-standby switch-over.

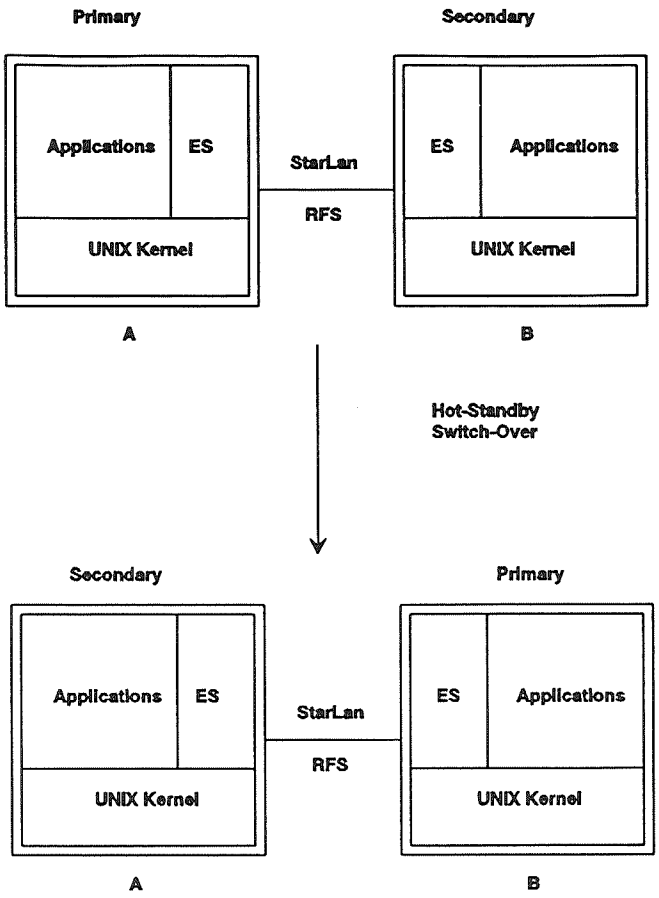

Figure 1. Command and Control Expert System Overview

Processor Self-Checking and Hardware Resource Management

Each ES monitors the health (such as disk space usage, CPU usage, and application process health) of the machine on which it resides. If hardware resource utilization reaches a certain threshold, the ES will first try to correct the situation by taking some actions that a system administrator would take under the same circumstances. If the ES detects problems that it cannot correct or problems that may cause system failure, it will give warnings to the operator. Human intervention is needed in this situation.

Hot-Standby Switch-Over

The two ESs exchange the health information of their host processors with each other. They also monitor the link between the two processors. If the ES on the primary processor detects potential fatal problems, it will send a signal to the secondary processor, and the secondary processor will take over the primary role. Because the switch occurs before the original primary processor failure, there is no system downtime.

The ES uses human heuristic rules to monitor, command, and control the hot-standby redundant system in real time. Unlike other commonly used fault tolerant systems, which take action after the failure occurs, this ES tries to predict failure and take action before failures occur.

HARDWARE AND SOFTWARE SELECTION

Why Use Expert Systems?

A processor usually has multiple processes running simultaneously, has many peripherals, and handles complicated communications. To ensure that the processor operates properly, and that tasks are performed successfully, the processor health status have to be monitored and adjusted continuously. Ideally, problems leading toward processor failure should be discovered and reported, if not solved, before the failure occurs. However, most processors cannot perform self-checking and error-correcting by themselves, and it is unrealistic to have human experts monitor and adjust the processors all the time. Therefore, in most applications, serious problems or even failures occur before human experts issue correction commands.

ES are suitable tools for performing processor monitoring, command, and control for the following reasons:

- Human experts use *rules of thumb* logic gained through experience to solve processor problems; ESs are the most effective tool (in comparison with other types of software) to capture the rules of thumb and use them to solve problems.
- The two research areas of this project, i.e., processor self-tuning and the hot-standby switching, do not necessarily need physical maneuvering. Control commands can be executed by software.
- The processor problems are rather complicated and therefore need specific knowledge and in-depth reasoning to be solved. Conventional programs are not equipped with the features to effectively capture and retrieve knowledge and reasoning.
- Experts agree on solutions. Domain knowledge can be translated into rules and relationships.
- Successful expert systems have been developed in similar applications [11].

Hardware Platform

The target processor for this project is the AT&T/6386, SCSI based workstation, running under UNIX SYSTEM V OS. Two workstations are connected through a StarLan network.

We chose this hardware platform because of its similarity to the ARINC Tower Data Link Service (TDLS). This would allow the results from this research effort to be easily used by TDLS or similar projects. Because UNIX is a machine-independent OS, the research results could also be applied to other UNIX platforms (such as SUN, HP) with little modification.

Software Development Tools

The ES shell chosen for this project is the Cxpert by Software Plus Ltd. Cxpert supports commonly used knowledge representation methods such as *Attributes*, *Frames*, *Procedures* and commonly used inference chaining methods such as *forward chaining* and *backward chaining*. Cxpert also provides query and window display facilities. In addition, Cxpert knowledge representation language (KRL) is fully compatible with C language.

UNIX PROCESSOR SELF-TUNING ES

Requirements Analysis

When application processes run on a UNIX processor, the hardware resources are used by these processes. The health of the processor affects the performance of the applications. For instance, when the CPU is overloaded, the response time of the application process is slower.

This ES prototype shall monitor and maintain the health of the processor in real time in three important areas: disk space, CPU load, and process management. It will solve and report problems in a manner similar to a system administrator, except that it can perform its task 24 hours a day continuously, while humans cannot (because it would be too expensive to be practical).

Knowledge Formalization

The self-tuning (or resource management) ES shall handle the situations stated in the *CONDITION* side of the following IF-THEN statements, and the action the ES takes is on the *ACTION* side of the statements. The following IF-THEN statements are called production rules. They are the body of the ES knowledge base (KB), and they are structured from human heuristics used to handle the same situations.

For disk space management, disk utilization in terms of disk usage percentage is checked every specified time period, as follows:

IF disk usage reaches a certain user-defined high water mark, and

> IF file archiving is necessary,
> THEN the ES will archive the specified files to tape and then remove them from the disk to create free disk space.
>
> IF the files were archived earlier,
> THEN only cleanup is performed.

IF disk usage increases at an abnormally fast rate,
THEN the ES will give the operator a warning.

IF disk usage reaches critical threshold (which means that after this threshold, any writing to the disk has a high probability of failure),
THEN the ES will announce that the processor is approaching a failure condition, and when switch-over is an option, it will initiate switch-over from the primary machine to the secondary machine.

The CPU load is monitored by the ES every specified time period, as follows:

IF the CPU is overloaded,
THEN, to improve system response time for critical processes, certain low priority processes will be terminated.

IF the CPU is still overloaded,
THEN the operator will get a warning.

In many airline or communications related applications, certain application processes must run continuously. The process management part of this ES runs checks every specified period of time:

IF the critical processors are not running,
THEN the critical processes are restarted by the ES automatically.

These are several of the simple but important scenarios that often occur on an applications processor. We studied the logic and procedures a system administrator would take under these scenarios and reformulated human knowledge into an *IF-THEN* format to facilitate ES implementation.

The ES can be expanded to handle more complicated and diversified problems, as long as the human heuristic for those problems can be structured into logical rules.

High-Level Designing

The high-level design for the resource management ES is illustrated in Figure 2. Each block depicts a logical function in the ES, and the arrow indicates data and its flow directions. This block diagram is applicable to each one of the three areas above.

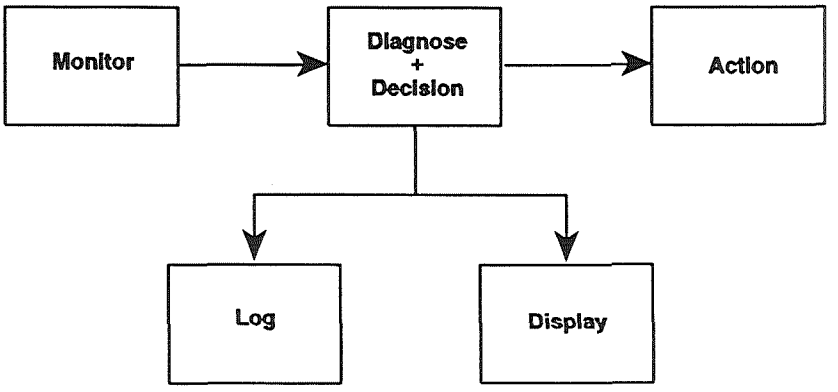

Figure 2. Self-Tuning ES Block Diagram

The *Monitor* (for each one of the areas above) uses UNIX system administration commands to collect resource utilization data. For example, the disk monitor function uses the *df* command to determine free disk space and other information about disk usage. *Monitor* then processes this data so that it can be used by the following processes depicted in the diagram:

- The *Diagnose and Decision* block is essentially a rule-based knowledge base that is formalized from human heuristic to diagnose the health of the resource utilization. This knowledge base fires different rules according to the information from the *Monitor*. The result of a fired rule is an action item, which selects and executes the proper action(s). This block also sends the action item into the log and display blocks.

- The *Action* block is a set of functions that will maintain the health of the processor resource usage.

- The *Log* saves the action item, time stamp, and brief health message onto disk storage.

- The *Display* displays health information and action items on a window system.

Implementation Details

For all three managed areas, the implementation is similar. *Monitor* and *Action* are implemented by C language, *Diagnose and Decision* and *Log* are implemented by Cxpert KRL, and *Display* is implemented by the Cxpert Hyperwindow system.

Performance Test

Test-run results show that the hardware resource management ES can handle all the fault scenarios described above and execute corrective actions regardless of the sequence of faults. The ES behavior is fully predictable. The one-round (i.e., a visit to the three targeted areas once) ES execution time is less than 15 seconds, which is much faster than comparable human action. A more sophisticated ES may have a longer one-round execution time, but it will still be much faster than manual operation. The performance speed can be improved by using C or C++ language for coding. In addition, operator error can be avoided because all the proper actions that the ES takes are pre-coded into the ES and tested.

HOT-STANDBY SWITCHING ES

Requirements Analysis

As shown in Figure 2, the hot-standby redundant system consists of two identical UNIX processors linked by StarLan Ethernet for peer-to-peer communications. When the system is first started up, one processor is designated to be primary and the other to be secondary. Hot-standby implies both machines are powered, and both are accepting and processing the same input. However, only the results of the primary processor are used as system output. Hot-standby switching requires that when the primary processor fails, the primary role is switched to the secondary processor within a specified period of time. This period of time must be short enough so that the *outside world* will not be affected by the processor failure.

Many mechanisms can be used to initiate hot-standby switching[7]. The ES uses the following mechanism:

> Each processor uses the self-tuning ES to check its own health and sends its health information to its peer processor via the StarLan link. If the primary processor detects its own failure, it stops its normal operation and sends a signal to the secondary processor. The secondary configures itself to be the primary processor and picks up the operation where the other machine stopped.

Knowledge Formalization

The production rules for the hot-standby switching ESs in the two machines are slightly different because of the different roles they play. The two processors use the self-tuning ES to check self-health and send this information to their peer processor via the link. In the meantime, each processor tries to read the health of the other machine.

In the primary machine-hosted ES, the rules are as follows:

IF the StarLan link is healthy, and

IF the secondary processor is healthy, and

IF the primary is going to fail (in the prototype, only disk problems are considered as fatal),
THEN the primary stops its output, announces its failure to the secondary, and sends out a switch-over command.

IF the secondary processor is NOT healthy,
THEN the primary processor will report that its peer is dead; no switch-over is allowed under this circumstance.

IF the link is dead,
THEN the primary machine will report this; no switch-over is allowed under this circumstance, either.

In the secondary processor, the rules are as follows:

IF the link is healthy, and

IF the secondary is healthy, and

IF the primary is dead, and

IF the primary sends a signal for switch-over,
THEN the secondary processor reconfigures itself to be primary and announces that it is primary now.

IF there is no switch signal from the primary,
THEN the secondary will announce this fact, but no switch-over occurs. (Note: this rule is intended to avoid a race condition between the two machines for the primary role and confuse or upset the *outside world*. A more elaborate rule set will allow the secondary to take over without the primary switch signal.)

IF the secondary is dead,
THEN it will notify the primary, and no switch signal will be sent to the secondary in case of primary failure.

IF the link is dead,
THEN no switch-over is allowed.

Listed above are some simple rules to control and coordinate the hot-standby switch-over process. More elaborate rules are necessary for a real operation context. These rules are for prototyping purposes only. The ES can be expanded to handle more complicated situations.

High-Level Design

Figure 3 illustrates the functional blocks in the hot-standby switching ES. The functionality of each block is explained by the block name. The ES high-level design is the same for both ESs, except that the primary machine has the *Send Switch* while the secondary machine has the *Receive Switch*.

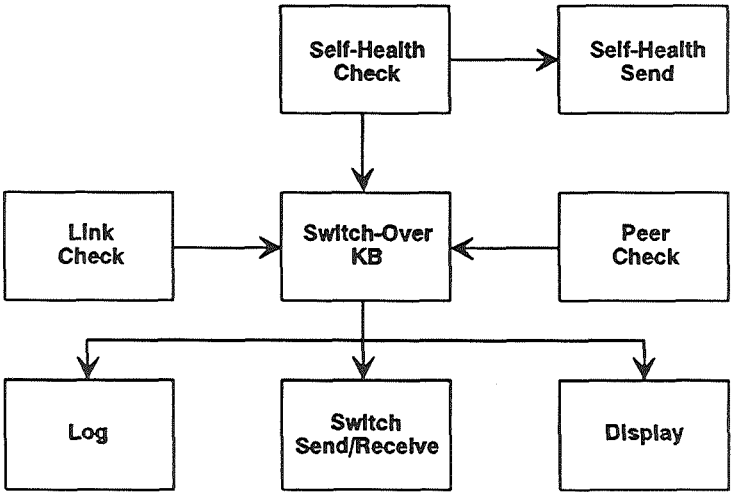

Figure 3. Hot-Standby Switching ES Block Diagram

Prototype Console Display

Figure 4 shows the primary and secondary machine console display. In the *DISK* window, machine disk space usage is given as a percentage. The ES action for self-tuning is also shown in the window. In the *CPU* window, average run queue size, average queue occupancy data, and the ES action for process control (such as start/kill process) are displayed. The *PEER* window shows weather the peer processor is healthy. The *LINK* window shows weather the RFS is still running on StarLan. The *SWITCH* window shows weather the host machine is primary or secondary. The *LOG* window logs the time, problem, and action taken by the ES.

There are also colors associated with the windows to indicate the status of the related area. If a critical situation occurs in a certain area, the corresponding window will turn red; if a warning occurs, the window will turn white, etc.

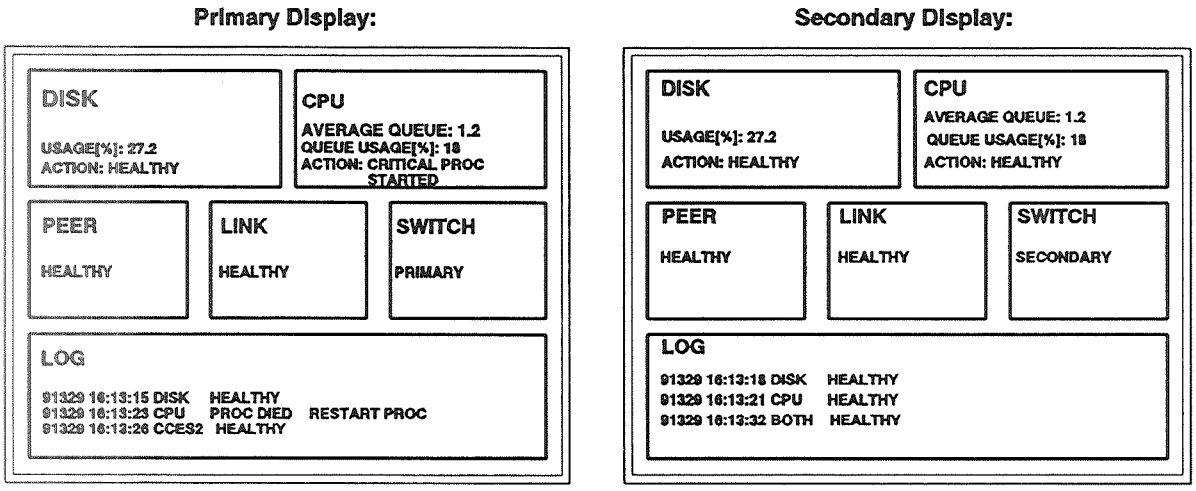

Figure 4. Prototype Console Display

Implementation Details

The *Switch-Over Knowledge Base (KB)* is implemented by Cxpert KRL, and the rest of the blocks are achieved by C functions.

The hot-standby switching ES incorporated the ES in the section titled "UNIX Processor Self-Tuning ES" for processor self-checking and self-tuning.

In this ES, in addition to the knowledge base, the technical challenge is the Inter-Process Communication (IPC). Regular IPC facilities such as message queues, unnamed pipes, and semaphores cannot communicate across the network, and the named pipes cannot satisfy the independency requirement between the two processors. This ES implementation uses a unique and innovative method that combines the StarLan Remote File Sharing (RFS) facility with file and record-locking techniques to accomplish the peer-to-peer communication across the network. This implementation will allow the two processors to operate independently if the link or one of the processors is dead. It will also allow the two processors to operate cooperatively if the link is healthy and both of the processors are running.

Performance Test

In addition to all the faulty scenarios presented in the section titled "UNIX Processor Self-Tuning ES", tests are generated to simulate link failure, primary failure, secondary failure, and both processor failures. The ES

performs correctly under these circumstances. The one-round execution time (which is the time that the ES takes to check the three self-tuning areas, the peer, and the link once) is less than 30 seconds, which means that the secondary will take over the primary function in less than 30 seconds in case of primary failure. A more sophisticated ES may have a longer one-round time, but performance speed can be improved by using C or C++ language to avoid Cxpert KRL overhead costs.

CONCLUSIONS

Command and Control Expert System (CCES) Potential Applications

The commercial application of the ARINC CC ES is very wide, because its knowledge base captures general UNIX system administration knowledge, and the rules that CCES uses are applicable to any UNIX system with little or no design change. Only the detailed implementation may vary for different platforms. For example, in the air traffic control industry, this ES has potential applications in a number of FAA directed services, the TDLS is one example.

Conclusions

The UNIX processor self-tuning ES prototype allows the processor to have better hardware resource management and, therefore, better performance and less chance of failure.

The hot-standby switching ES prototype provides coordination, command, and control to the switch-over process and, therefore, reduces system downtime and improves system reliability and availability.

REFERENCES

[1] A Guide to Expert Systems, Waterman, Addison-Wesley, 1986

[2] "Applying Systems Analysis Techniques to Knowledge Engineering", Expert Systems, Vol. 7, No. 2, May 1990, Swaffield, G. and Knight, B.

[3] "Architecture of Fault-Tolerant Computers, IEEE Computer, Siewiorek, D. P., August 1984

[4] "Artificial Intelligence Technologies for Real-Time and Object-Oriented Applications," Electrical Communication, Vol. 62, No. 3/4, Barachini, F., 1988

[5] AT&T StarGROUP Software Reference Set, AT&T, 1988

[6] Building Expert Systems, Hayes-Roth and Waterman, Addison-Wesley, 1983

[7] Design and Analysis of Fault Tolerant Digital Systems, Johnson, J. W., Addison-Wesley, 1989

[8] "Design for Ultrahigh Availability: The Unix RTR Operating System," IEEE Computer, Wallace, J. J. and Barnes, W. W., AT&T Bell Laboratories, August 1984

[9] "Expert Systems Making Quiet Inroads into Networking Applications, " Networking Management, May 1991

[10] "The Real-Time Expert," BYTE, Laffet, T.J., January 1991

[11] "YES/MVS: A Continuous Real Time Expert System, " Proceedings AAAI-84, Griesmer, J. H., <*et al*>; IBM, 1984

[12] "New Controls for Air Traffic," IEEE Spectrum, February 1991