Computer System Performance Problem Detection Using Time Series Models

Peter Hoogenboom and Jay Lepreau

University of Utah

Abstract

Computer systems require monitoring to detect performance anomalies such as runaway processes, but problem detection and diagnosis is a complex task requiring skilled attention. Although human attention was never ideal for this task, as networks of computers grow larger and their interactions more complex, it falls far short. Existing computer-aided management systems require the administrator manually to specify fixed "trouble" thresholds. In this paper we report on an expert system that automatically sets thresholds, and detects and diagnoses performance problems on a network of Unix computers. Key to the success and scalability of this system are the time series models we developed to model the variations in workload on each host. Analysis of the load average records of 50 machines yielded models which show, for workstations with simulated problem injection, false positive and negative rates of less than 1%. The server machines most difficult to model still gave average false positive/negative rates of only 6%/32%. Observed values exceeding the expected range for a particular host cause the expert system to focus on that machine. There it applies tools with finer resolution and more discrimination, including per-command profiles gleaned from process accounting records. It makes one of 18 specific diagnoses and notifies the administrator, and optionally the user.[1]

1 Introduction

Existing computer and network management tools require the administrator manually to specify fixed threshold values of performance or load criteria. This paper describes a time series modeling technique that can be used for more automatically detecting computer system performance problems. The technique is based on an *exponentially weighted moving average* time series model. It allows detection of performance problems in a host by providing a means of detecting performance criteria values that are out of normal ranges. The effectiveness of the technique is demonstrated by its use in the **System Performance Advisor** (SPA) system administration expert system. The purpose of SPA is to assist a Unix[2] system administrator in *system performance management.* We define system performance management to be those activities performed by a system administrator to ensure a computer system is providing as much of its capacity as possible to its users. SPA does this by monitoring hosts and processes in a network. Problems with hosts and processes make themselves apparent as a deviation from normal activity levels. After a problem is detected, SPA alerts the system administrator.

Performance can be defined and measured in many ways. Typically, the system administrator chooses system metrics that are easily obtained through standard utilities. Measurements such as CPU utilization, memory utilization, paging rate, and load average are commonly used.

The process of managing a system's performance is an iterative one. Performed by a human, the steps involved might not be as well-defined as described here, but the basic iterative steps still exist. The performance management process is composed of a *definition* phase, a *diagnosis* phase and a *therapy*

[1] This research was supported in part by the Hewlett-Packard Research Grants Program and the University of Utah.

[2] Unix is a trademark of AT&T Laboratories.

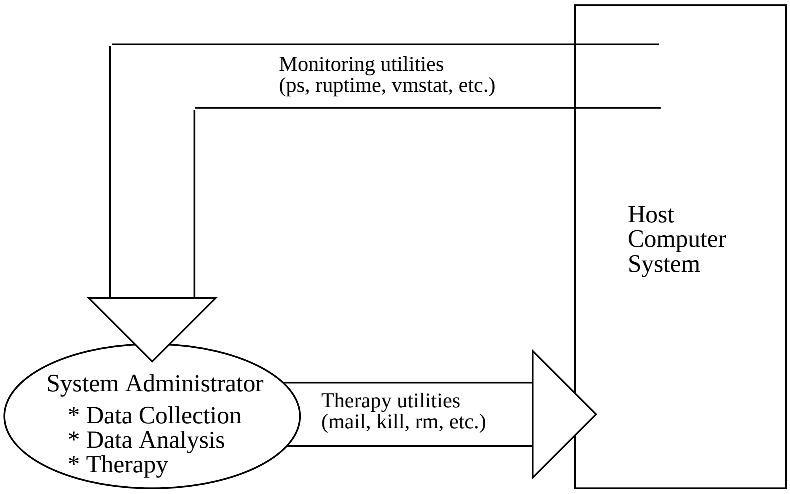

Figure 1: The Traditional Process of System Performance Management

phase. The definition phase consists of determining what performance criteria are to be evaluated and how the data will be collected and analyzed. Diagnosis begins by making measurements of the system. This is followed by analysis of the collected data. Analysis is done manually or with assistance from another computer. During analysis, the performance problems are recognized and categorized. After the analysis, if the results suggest that some improvements in performance are possible, a therapy phase is begun. During this phase, adjustments to the system are made in an attempt to alleviate the problem. The remaining phases are repeated until there are no known performance problems in the system.

2 Why is System Performance Management Difficult?

Traditionally, system performance management activities are carried out by human system administrators who rely on their own expertise to ensure maximum performance of all systems. As shown in Figure 1, the system administrator monitors the host computer system, analyzes the collected data, and performs a therapy procedure on the host as needed. In addition to this process being very time-consuming, there are several difficulties with this approach [13]:

- frequently, the best guesses by system administrators are wrong; and
- performance degradations of up to 20% go unnoticed on a regular basis.

In addition, regardless of the problem domain, human expertise suffers from several other limitations:

- it is not always available (illnesses, vacations, etc.);
- it suffers from inconsistencies due to skill level, emotional state, and attentiveness; and
- it is not scalable. For example, suppose a human system administrator can consistently and objectively manage the system performance of all users and programs on 25 processors. Can the same system administrator effectively manage all users and programs on 250 processors?

An expert system approach to managing system performance (shown in Figure 2) can reduce or eliminate these disadvantages. It is always available, it applies its expertise consistently, and it is scalable. Since the human's expertise relies on remembering what a given host, user, or program has done in the past, the more hosts, users, and programs he has to deal with, the less knowledge he has on an individual host, user, or program. This does not happen with an expert system approach.

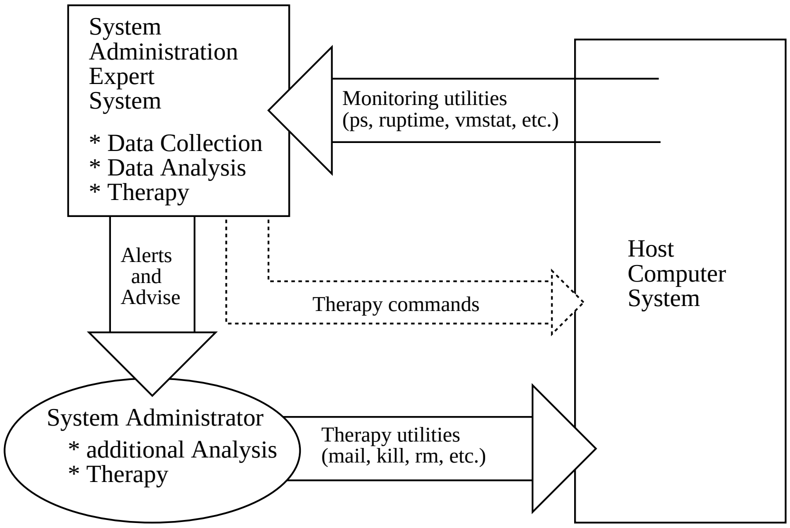

Figure 2: The Expert System-based Process of System Performance Management

3 Developing a Time Series Model of Computer System Performance

Before developing the SPA expert system, we found it necessary to define a mechanism that could reliably indicate normal system performance and the expected amounts of deviation from that performance level. Setting fixed threshold levels for problem detection, as today's commercial network management systems require, was deemed unacceptable. here are several realities of computer system usage that make it difficult to define a single, static model of "expected" behavior that is appropriate for all hosts in the network, or even setting a different one for each host (tedious as that might be):

- Each processor (host) can have a speed and configuration different from other processors on the network.
- The expected workload distribution is different on each processor.
- The expected workload distribution on each processor varies over time. Variations can occur daily, weekly, or over longer periods of time.
- Each user on the system wants to accomplish different types of tasks. Users that use the same program do so in a way that could be unique to the user.

For these reasons, SPA must maintain a different model of the expected utilizations and activity levels for each host. Several possible system utilization modeling techniques were investigated.

3.1 Modeling Alternatives

The purpose of maintaining a model of hosts and processes is to be able to, at time t, make a forecast of what the behavior will be at time $t+1$. If the forecast error (the difference between the forecast and the actual value at time $t+1$) is significant, there is a performance problem to be investigated.

There are several ways to model hosts and processes. The modeling techniques discussed below are taken from different branches of mathematics, but all have been extensively studied. This provides a rich mathematical background on which to base the modeling technique chosen for SPA.

The successful modeling technique must satisfy the following criteria:

Forecast Quality: The model must be able to accurately predict future behavior. No modeling technique is able to exactly predict future behavior, but some modeling techniques are better at modeling certain kinds of behavior. A modeling technique should be chosen to minimize forecast errors as much as possible.

Time Efficiency: The model provides a simple way to determine the forecast, the forecast error, and whether the forecast error is significant. The model provides a "best guess" at time t of the behavior at a future time $t+1$.

Space Efficiency: The model does not require maintaining an extensive amount of history information. Less history data means less space used to store the model, but some history data is probably required to maintain forecast quality.

Responsiveness: It is able to deal with changes in the characteristics of the data being modeled. Much of the data being considered undergo some kind of seasonal variation. The model should be able to modify itself automatically to adjust to these changes. If the user were required to make these adjustments on all the hosts and processes being modeled, the models would quickly become out-of-phase with the actual behavior, because the user probably could not keep up with the necessary changes.

Configurability: If necessary, the system administrator should be able to modify the behavior of the model to improve its performance. This is also known as *tuning* the model.

Interpolation Techniques

The data collected by SPA are implicitly a function of time. For example, suppose there exists a series of n values $y_1, y_2, \ldots, y_n$ collected at n distinct times $t_1, t_2, \ldots, t_n$. The collected values can be described by a function $Y(t)$. Thus,

$$y_1 = Y(t_1), y_2 = Y(t_2), \ldots, y_n = Y(t_n).$$

The model should be able to determine what value should be expected at time t_{n+1}. That is, what is $Y(t_{n+1})$? If a function $Y(t)$ can be determined that completely describes the data collected so far, it can be used to find out $Y(t_{n+1})$. It can be shown [14] that given n data points, a unique polynomial of degree $n-1$ can be found which has the desired property of satisfying the equation shown above.

Polynomial Interpolation and *spline interpolation* are common approaches to this. Both techniques suffer from similar problems that reduce their likelihood of being used for system performance analysis. First, interpolation is very effective in fitting a curve to known data points, but not so effective for forecasting a future reading, even if the time at which the reading takes place is very close. Predictive abilities are worse when the data points are partly stochastic in nature. Second, the mathematics for even one model is complex. Polynomial interpolation is $N \cdot (\log N)^2$ for advanced methods and quadratic for simpler techniques [14]. Although a cubic spline on n points can be constructed in linear time, few, if any, computer system performance criteria are modeled accurately with a cubic equation. Third, extensive history information is required. One could reduce the amount of required history by only using the last N readings, but this is at the cost of forecast quality.

Curve Fitting

This approach is closely related to that of interpolation discussed in the previous section. The difference with this approach is the admittance of a stochastic element in the data. Thus, $Y(t)$ does not exactly describe all the points $y_1, y_2, \ldots, y_n$. However, the "best" (least amount of error) $Y(t)$ is still desired. Numerous methods of minimum discrepancy exist to fit data to a linear function (commonly known as *the method of least squares*) and more complex functions. Periodic functions such as those necessary to model daily, weekly, and quarterly variations in workload on a computer system can be approximated accurately using fast Fourier transform techniques.

Unfortunately, the mathematics is complicated and time-consuming for techniques more advanced than least squares fit for a linear function. The network in which SPA operates has 100–200 hosts that are used by hundreds of users. There are dozens of supported programs on each of these hosts. To implement models using these techniques does not allow this level of scalability. Another disadvantage is the slowness of response to changes: no allowances for a seasonal variation are made.

Queueing Theory

Queueing theory is a branch of mathematics that deals with situations involving waiting for access to a shared resource. There are many applications for a theory of queues including factory assembly lines, highway traffic flow, jet traffic near an airport, a checkout counter at a supermarket, and, not surprisingly, computer systems.

Discussions of queueing theory usually refer to the objects waiting in line as *customers* and to what they are waiting for as a *service center*. The description and analysis of arrivals of customers and the delay they encounter waiting for service is the object of study in queueing theory. Typical quantities that are provided from a queueing model are the average number of customers in the queue and the average time a customer spends in the service center.

One attractive feature of a queueing model is that it provides a modeling technique in which the model is expressed in terms similar to the problem domain. Another attractive feature is that the use of queueing theory for computer system performance evaluation has already been extensively studied and reported. Most notably are the books by Ferrari, Serazzi, and Zeigner [4] and Lazowska, Zahorjan, Graham, and Sevcik [7].

There are several problems with using queueing models for the purpose of real-time problem detection. First, it is not clear from the queueing theory literature how one determines the parameters of the queueing model from historical data. Second, after they are initialized, a queueing model is not responsive to changes in the characterizations of workload. Third, after the model is in use, updating its parameters to reflect a changed workload must be done manually.

Probability Distributions

Probability distributions are employed when the variable of interest is continuous or nearly continuous. For a continuous variable, the probability of encountering any one value is of little interest because the probability is zero when the number of possible values is infinite. Thus, the probability of a variable being in a certain range of values is more interesting. Figure 3 shows a typical frequency distribution of a time series for load average measurements. Detecting problems using this type of model is a matter of deciding that the probability of a given value occurring is low enough to be considered unusual.

In the case of Figure 3, the probability of a load average measurement being higher than, for example, 10.0 can be calculated. The historical data for the host provide a total number of samples taken and how many of those are values above 10.0. From this information, the probability of an individual load average being above 10.0 can be calculated.

There are several disadvantages to this approach. First, a large number of samples must be collected and reviewed in order to determine a probability distribution that is reasonably accurate. Second, in order to account for workload variations, the distribution must vary over time. This could be approximated by having a different distribution for each hour of the day. To account for daily and weekly variations on N hosts requires $N \cdot 7 \cdot 24$ distributions. Third, the range of values possible and the number of intervals must be known beforehand. If not, the system must dynamically recalculate the distribution ranges, interval count, and probabilities. This could be computationally expensive if the recalculation occurs frequently.

Time Series Models

Time series analysis is based on the assumption that the data collected varies as a function of time. This is certainly true for load average measurements. The idea behind time series analysis is to examine the

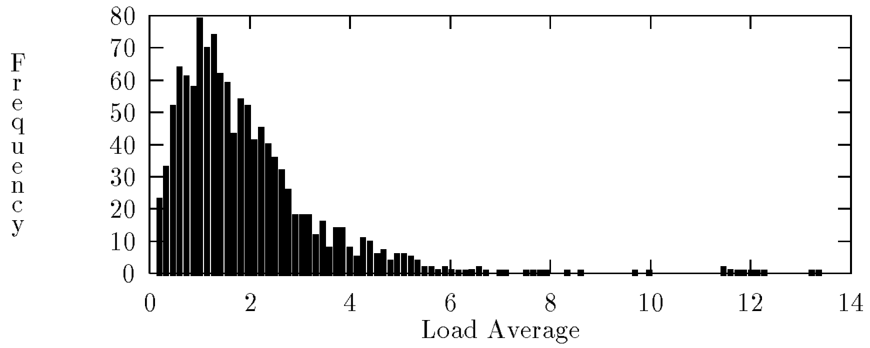

Figure 3: Typical Load Average Frequency Distribution

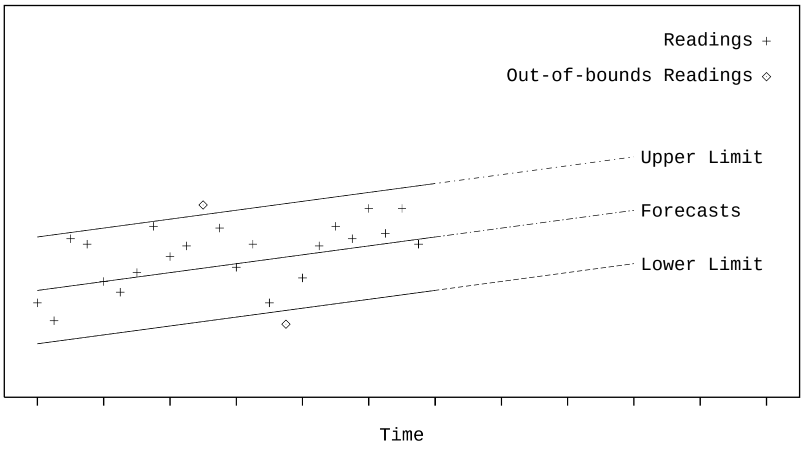

Figure 4: A Linear Time Series Forecasting Model

history of data collected and do one or more of three things: recognize patterns in the history data, calculate statistical measures of the history data, or forecast what the time series will do at some future time.

Figure 4 shows a forecasting model that recognizes readings as abnormal when they are more than a fixed amount from the forecast. In this example, the forecasts are described by the equation of a line, but could also be the current mean of the readings. The upper and lower limits could be one standard deviation from the mean.

By maintaining a history of readings collected from the system, useful statistical measures such as mean, standard deviation, minimum, and maximum can be calculated. This type of time series model is called a *constant model* because it can be expressed algorithmically as

$$x_t = m + \epsilon_t$$

where x_t is the forecasted value at time t, m is the mean, and ϵ_t is a random component that is assumed to have mean 0 and variance v_ϵ. One difficulty with this model is dealing with variation in the modeled object over time. The more readings the history contains, the slower the mean responds to changes in the process and the more costly it is to calculate.

One way to speed the response and calculation of the mean is by using a *moving average of length* k [11]. The moving average is calculated by considering the last k readings only. The moving average model does handle a seasonal variation in the data; however, it requires a fixed number of readings in each season to accomplish this. Also, if the season is long, a large number of readings must still be kept to calculate the moving average.

For data values that are known to experience seasonal variations, a *seasonal model* can be employed. The seasonal model is also known as an *exponentially weighted moving average* time series model [17]. Seasonal models arise frequently in time series analyses of business and economic data. In those types of studies, the readings often fluctuate according to the seasons and business cycles. Often it is desirable to remove the effect of these fluctuations to study the readings without the influence of a particular season. Other times, a forecast of what to expect in the next time period is sought. An accurate forecast requires a model that can account for the seasonal variation. The seasonal model also overcomes the deficiencies of the moving average model: any number of readings can be taken and once the model is initialized, no history data values are required.

In a seasonal time series model, the values being modeled have four components: *constant*, *trend*, *seasonal*, and *random*. The model can be used to account for the first three of these components. The constant component is the portion of the data that is always present. The trend component reflects the fluctuation in the data that extends throughout the entire time series. For example, the utilization of a computer system might increase from one year to the next as more people use it. The seasonal component reflects the regular variations that occur over a specific period of time. For example, the daily variation in workload readings. Usually, they are higher during the day than at night. Finally, the random component accounts for fluctuations in the data due to undetectable causes.

The basic form of the seasonal time series model is written as

$$x_t = b_1 + b_2 t + c_t + \epsilon_t$$

where b_1 is a constant component, b_2 is a trend component, c_t is a seasonal factor, and ϵ_t is a random error component. The effect of the seasonal factors c_t is to de-seasonalize the current reading x_t. The length of the seasonal variation is fixed at length L. In the case of a load average time series for a host, $L = 24$ hours.

Because b_1, b_2, and $c_t, t = 1, \ldots, L$ cannot be determined exactly, they must be estimated. These estimates are updated at the end of each of the L periods. Thus, these estimates respond quickly to any changes in the profile of the data.

The model adapts to changes in the data by the use of three *smoothing constants*: α, β, and γ. The usage of these smoothing constants is analogous to the usage of the decay rate in the calculation of load averages in Unix operating systems [8]. The α, β, and γ smoothing constants are used to smooth the constant, trend, and seasonal components of the time series model, respectively.

A forecast of a data value at some time $t + k$ in the future is computed from:

$$x_{t+k} = \hat{b}_1(t) + \hat{b}_2(t) \cdot k + \hat{c}_{t+k}(t + k - L)$$

where $\hat{b}_1(t)$ and $\hat{b}_2(t)$ are the estimates of $b_1(t)$ and $b_2(t)$ at the end of the time period t, and $\hat{c}_{t+k}(t+k-L)$ is the estimate of the seasonal factor for period $t + k$ that was computed L periods ago.

The estimates $\hat{b}_1$, $\hat{b}_2$, and $\hat{c}_t, t = 1, \ldots, L$ are computed as follows:

$$\begin{aligned}
\hat{b}_1(t) &= \alpha\left[x_t - \hat{c}_t(t - L)\right] &&+ (1-\alpha)\left[\hat{b}_1(t-1) + \hat{b}_2(t-1)\right] \\
\hat{b}_2(t) &= \beta\left[\hat{b}_1(t) - \hat{b}_1(t-1)\right] &&+ (1-\beta)\left[\hat{b}_2(t-1)\right] \\
\hat{c}_t(t) &= \gamma\left[x_t - \hat{b}_1(t)\right] &&+ (1-\gamma)\left[\hat{c}_t(t-L)\right]
\end{aligned}$$

where $0 < \alpha, \beta, \gamma < 1$.

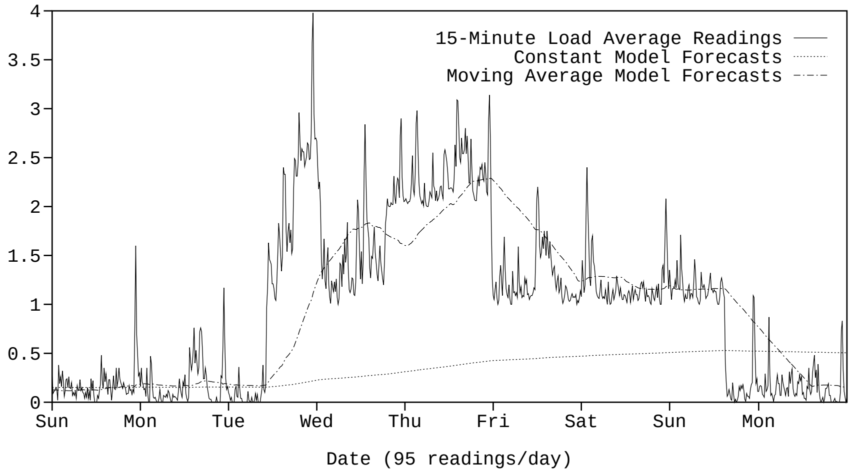

Figure 5: Constant Model and Moving Average Model of 15-Minute Load Average

The calculation of the initial estimates $\hat{b}_1(0)$, $\hat{b}_2(0)$, and $\hat{c}_t, t = 1, \ldots, L$ can be done from historical data using a least-squares linear regression [11], or by simpler methods[9]. After the initial estimates are computed, the model is used with no further need to reference the historical data. The smoothing constants α, β, and γ are determined heuristically by the model developer. Smaller values for the smoothing constants give more weight to previous readings. This results in the model responding more slowly to changes in the time series. Larger values give more weight to recent values. Thus, the model reacts more quickly to changes. Very large values are to be avoided, however, since the model will over-react to random fluctuations.

The seasonal model described above is one of a family of seasonal models based on a method originally described by Winters [17]. The model and notation described above is from Montgomery, Johnson, and Gardiner [11].

Comparison of the Three Time Series Models

Three varieties of time series models have been described. How well the model performs depends on the values being modeled. An example of the usage of these three models on a series of 15-minute load average readings from `jensen.cs.utah.edu` is given in Figures 5 and 6. The constant model and moving average model (length is 95 readings) are shown in Figure 5. The seasonal time series model is shown in Figure 6. This host is a HP 9000/380 with 32 MB of real memory. It serves mainly as a fileserver, but also receives moderate usage interactively during the day and in compiling large software systems at night. The displayed values are one portion of 3 weeks of load averages collected. This portion of the data shows the host experiencing long periods of time in which the load averages are always above 1.0. The constant model is clearly unresponsive to the changes in the workload characterization. This is due to the long past history of readings. The moving average model provides much better response: forecasts errors are within tolerable limits within a day. The time series model provides even better response. The standard deviation of forecast errors was 0.6716 for the constant model, 0.4316 for the moving average model and 0.1077 for the seasonal model.

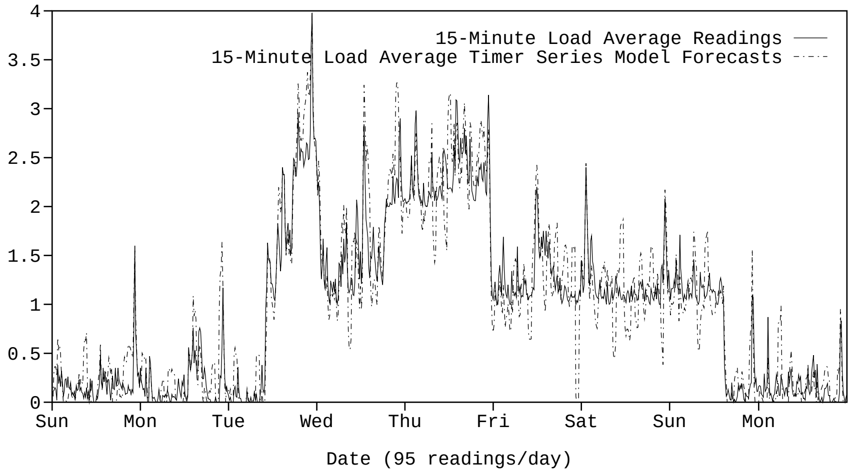

Figure 6: Seasonal Time Series Model of 15-Minute Load Average

4 The SPA Expert System

The SPA expert system was developed as a means of validating the use of time series models in detecting computer system performance problems. It consists of 10,800 lines of Common Lisp code and two small C language programs. The development of the Common Lisp portion of SPA, hereafter referred to simply as SPA, was accomplished using the FROBS system [12] as an expert system shell. This system provides a merger of two common knowledge representation techniques: object-oriented programming and frame-based data structures. The programmer can choose the technique that seems most suited to the problem at hand. In the case of SPA, FROBS forward-chaining rules and frame data structures[3] can be chosen to express data-driven reasoning about hosts and processes or object-oriented programming techniques can be chosen to express data abstraction and build hierarchies of objects in the knowledge base.

The forward-chaining rules provide SPA the ability to reason and make decisions about the knowledge base. A forward-chaining rule can be thought of as an IF *condition* $THEN$ *action* statement. If *condition* (also called the *premise*) is true in the knowledge base, then the rule is said to *fire*, and the specified *action* (also called the *conclusion* of the rule) is performed. The specified *action* can have side-effects in the knowledge base which might cause the firing of additional forward-chaining rules.

SPA's forward-chaining rules describe a relationship between what is currently true about the knowledge base and what actions SPA should take. This, along with the ability to dynamically add and remove rules from the system give SPA a considerable advantage over system administration tools written in a compiled, procedural language. Experience with expert systems has shown it normally impractical to encode and tune the large required knowledge bases, when expressed strictly procedurally.

A functional description of the SPA expert system and its major software components is shown in Figure 7. The Common Lisp code contains the knowledge base, inference engine, user interface, the SPA software monitor (SPASM), and the SPA mail interpreter (SPAMI). SPASM is responsible for collecting data from the hosts in the network and asserting this data in the appropriate knowledge base objects. SPAMI is responsible for sending mail to users and the system administrator when SPA suspects that problems exist. It is also responsible for receiving mail from users regarding processes that might be involved in performance problems. The two C language programs (HOSTHIST and PROGHIST) are used to collect host and process data at regular intervals even if SPA is not running. This is needed to keep the time series models up-to-date. Currently, HOSTHIST runs once every 10 minutes and collects load average

[3] A single instance of these is called a *FROB*.

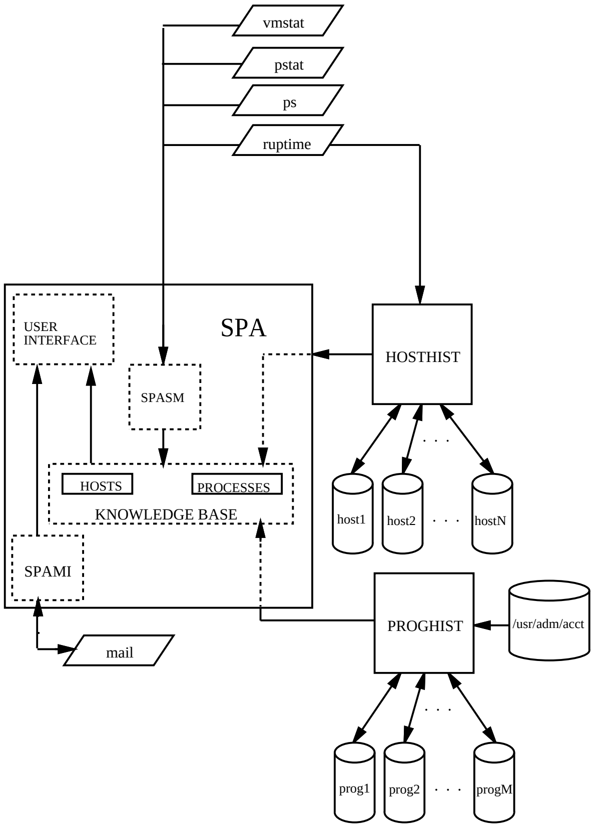

Figure 7: SPA Functional Block Diagram

and user count data from the `rwhod` database. PROGHIST is set up to run once every day and collects resource utilization information from system accounting records. SPA runs when the system administrator requests it.

The SPA system interfaces with the host operating system by invoking standard utilities and a System Administration Tools [15] (SAT) dynamic relation to collect data about hosts and processes in the network. Standard utilities such as `vmstat`, `pstat`, and `ruptime` collect host-level data. The SAT relation (called `kmem`) provides a means to collect process data on any host in the network. Data that are collected by the system are stored in the SPA knowledge base. The set of forward-chaining rules, heuristics, and time series models contained in the SPA system are used to examine the knowledge base and determine if a problem exists. After a problem is found, another set of rules is used to decide what action should be taken.

For each performance criterion that SPA monitors, the time series models used for problem detection are initialized from a history of readings. After the model is initialized, readings are gathered by SPA and compared to an expected value determined by the model. Readings that are not within the threshold range result in the creation of an alert that is displayed for the system administrator. The system administrator is responsible for making a determination of the validity of the alert. Validation can be accomplished by examination of the knowledge base, collection of additional data, or by running additional system utilities.

4.1 Problem Detection Within the Network

SPA first examines a host's load average to determine whether the host is involved in a performance problem of some kind. To do this, SPA retrieves the data maintained by the `rwhod` program and diagnoses one of the problems types listed in Table 1.

Table 1: Description of SPA Host-level Problems

Problem Type	Description
`status`	the host is down
`fileserver_down`	a host used as a fileserver is down
`load-high`	load averages are higher than expected
`user-count-high`	number of users on the system is higher than expected
`wedged-process`	1, 5, and 15-minute load averages are identical and non-zero

As described in the previous sections, SPA uses a time series model of a host's load average readings to determine when a load average reading is abnormal. However, the system administrator can specify absolute limits as thresholds for load average problems. Any load average readings over the upper threshold will always cause creation of a problem instance. Likewise, any load average reading under the lower threshold will never cause creation of a problem instance. This approach is similar to the setting of threshold limits that is available today in many commercial system management tools.

The `wedged-process` problem type shows interesting heuristics, and turns out to be very useful in practice. It is diagnosed when there is a host for which all three current load average readings (1, 5, and 15 minute) are the same, non-zero, integral value. This condition is used to detect processes which are hung, or looping in some way on a host that is otherwise little used at the time. This problem type shows up frequently when SPA is first run in the morning: processes that ran overnight and did not complete for some reason will cause the load average to settle in to a non-zero, integral value.

Another interesting rule is `nfsd-processes`. This problem is diagnosed when the load average is "pegged" at the number of `nfsd` processes found. It typically occurs when a naive user runs "find" over our global "root" directory.

4.2 Problem Detection Within a Host

After a host is discovered with a workload that exceeds the threshold value, SPA applies tools with finer resolution and more discrimination, such as the `pstat` and `vmstat` commands and the `kmem` SAT dynamic relation described above. The data about the processes involved in the current workload is compared to per-command profiles gleaned from process accounting records. Again, if there is a process whose data values are found to exceed expected ranges, that process is treated as a source of the performance problem on that host. At the process level, SPA will detect the problem types described in Table 2.

4.3 Interacting With SPA

The system administrator has commands available to inspect the knowledge base, perform actions on the knowledge base and the hosts and processes in the network, and to query SPA about its operation.

Inspecting the Knowledge Base

The `show`, `select`, and `plot` commands are the primary means by which the system administrator inspects the knowledge base. The `show` command takes any number of knowledge base object names as arguments and displays the contents of those knowledge base objects.

The limitations of this approach to inspecting the knowledge base were quickly made evident when SPA was run on a large number of hosts. When there are thousands of knowledge base objects, frequently the system administrator does not know the names of the desired objects, only the qualities of such objects. Thus, a `select` command was implemented which allows querying the knowledge base in the fashion of a relational database.

Table 2: Description of SPA Process-level Problems

Alert Type	Description
kernel_table	one or more kernel tables is approaching 100% used
mem_size	the memory size of a process is larger than expected
time_used	the CPU time used by a process is larger than expected
elapsed_time	the elapsed (wall) time since a process started is larger than expected
cpu-utilization	process or user is using more CPU resources than expected
mem-utilization	process or user is using more memory resources than expected
nfsd_processes	many nfsd processes with a high rate of CPU utilization
scan_rate_problem	the paging algorithm is scanning at a higher than expected rate
abandoned-process	a process has no parent process
reniced-process	a process has been reniced
zombie_process	a process in its final exit state remains on the system
paging_alert, scan_rate_alert	a host's processes are consuming enough memory to cause paging

The user specifies what kinds of knowledge base objects to search (e.g., `host`, `process`, `problem`), which slots of those objects to display, and a search condition. The search condition specifies a relation that must evaluate to true for each knowledge base object that is a member of the displayed result. The `select` command has the most elaborate syntax and semantics of all the SPA commands: it is implemented as a subset of the SQL [1] relational database query language. A formal semantics of the SPA `select` command is given in [6].

After the `select` command is parsed, a FROBS rule is defined. The specified search condition becomes the premise of the rule. The conclusion of the rule contains a call to an auxiliary function that displays the requested slots of the knowledge base objects that caused the rule to fire. After the rule is defined, the inference engine is allowed to run to determine whether there are any knowledge base objects for which the search condition is true. After the rule has fired for all knowledge base objects, the rule is deleted.

In addition to looking at the current readings in the knowledge base, the system administrator frequently wants to look at the past behavior of a host or process. This can be done with the `show history` command or the `plot` command. The `plot` command uses the `gnuplot` utility to provide a graphical representation of the history of readings.

Performing Actions With the Knowledge Base

The `continue` command is the primary means by which the user initiates the process of data collection, data analysis, and problem detection. Once this command is issued, SPA continues this process until one or more problems are detected via the time series models.

When a problem is detected, the system administrator can enter a therapy phase in which SPA guides the system administrator step-by-step through a procedure script to resolve the problem. This is done with the `advise` command. At each step, the system administrator can have SPA do the step, skip the step, perform the step manually, or explain why SPA is recommending this action. These scripts are written in a language that is a subset of Common Lisp with extensions to run Unix commands, provide explanations, and access the knowledge base.

Querying SPA About Its Operation

SPA provides several facilities to assist the system administrator while using the SPA system. The `help` command provides a short description of commands, data collection types, and global variables in the SPA system. The `why` and `diagnose` commands allow the system administrator to ask SPA to explain a problem diagnosis. The ability to provide explanations of its reasoning and diagnoses is an essential feature of an expert system. Without it, the system administrator never learns to trust the findings of the expert system. In SPA, explanations come in several forms:

- how the system reached a conclusion,
- a recommendation on what to do to obtain more information about the current problem,
- references to other documentation for further information on this kind of problem, and
- suggested solutions (if any) to resolve the problem command.

4.4 Customizing the Behavior of SPA (Extensibility)

One feature of SPA that is important to its usefulness is that it is *extensible*. If system administrators find that there are additional problems that they would like SPA to detect, a new knowledge base rule can be written. The rule is built by specifying the conditions that must be true when the problem is in effect. A time limit for detection of the new problem type can also be specified. For example,

```
include from process \
where  user = 'hoogen' and ppid = 1 and ( mem > 10.5 or cpu >= 50.0 ) \
as big_orphan
```

creates an instance of a problem whenever it finds a process owned by the user `hoogen` that has `init` (pid = 1) as its parent and is consuming either more than 10.5% of real memory or at least 50% of the CPU cycles. This 3-line `include` statement is translated into a FROBS forward-chaining rule containing 25 lines of Common Lisp code. This rule is checked during SPA's problem analysis phase to see if the conditions have been satisfied in the knowledge base. If so, an instance of the new problem type (e.g. `big_orphan`) is created and displayed.

5 Results

5.1 Time Series Model Validation

We validated the time series modeling approach on one type of data, the load average. This validation could be done on many other types of data, but load average was the most practical for us to collect. Its excellent results suggest that other measures of resource use would also respond well to time series modeling, but those analyses were not performed.

The time series modeling approach was validated by recording the load average readings for 50 hosts. By type of use, 40 can be classified as workstations, and 10 as servers. In our environment, as in most current large installations, the vast majority of machines are workstations. They are either "desk" machines dedicated to a particular person, or "lab" machines used by different people, but usually only one at a time. For this study, however, we selected a disproportionate number of fileservers and general use machines, since we expected, correctly, that they would be more difficult to model accurately.

An assumption underlying this analysis is that the observed loads were mostly "normal," e.g., not the result of looping processes. This was almost certainly true, but if it were not, the effect is that our model will be more accurate than predicted from the analysis. It should not significantly affect the actual values we derived.

The 15-minute load averages were collected every 15 minutes for three weeks, yielding 2100 readings for each host. Each host's records were separately analyzed to gain a "feel" for the data. This consisted of searching for the values of six different variables that would minimize an evaluation function. Three variables are the three smoothing constants. A fourth is related to the standard deviation of the model's forecast *error*: a multiple of that standard deviation is chosen to be the threshold for problem detection.

A fifth is a heuristic, based on the semantics of load average, which modifies the threshold to be at least T greater than the model forecast. T is typically in the range of 0.8. The sixth variable is the value chosen for the simulated problem reading.

The evaluation function is a measure of the false positive reports the actual data would generate (i.e., when a data point was significantly higher than the model's prediction), combined with a measure of false negatives. The latter is estimated at each of the 2100 points by pretending ("injecting") a load L higher than actually recorded, and recording whether the model would have flagged it or not. We tried values of L from 0.5 to 1.0, based on the effect of most real-world problems on load average.

Our results showed that:

- Workstations could be modeled precisely, with near perfect accuracy of problem detection. For $L = 1.0$, both false positive and negative rates were substantially less than 1%.
- Servers, however, were more difficult to model accurately, as shown in Table 3. This stems not from additional variance on the servers, but from the higher average load. Their absolute variance is higher, but the injected problem load (L) is an additive constant, the same value as that for workstations. This stems from our attempting to discover even one "runaway" process. This may be an overly stringent requirement.
- Although not as accurate for servers, the models still works well enough to be very useful.
- For almost all hosts, both workstations and servers, the same values of the smoothing constants are optimal. These are α (0.9), β (0.1), and γ (0.2). A few were 1 or 2/10th's different, but this shows that it's generally not worth the trouble to tune these constants on a per-host basis. Winters [17] says that the response of this type of time series model to changes in α, β, and γ is "flat": that is, small changes in the parameters have small effects on the quality of the forecasts. The analysis of the models tested here supports that theory. However, large deviations from the optimal values of the constants did result in substantially worse predictions.
- For workstations, a nearly optimal threshold for problem detection was 2.0 standard deviations above the mean forecast error. Modifying the threshold to be at least T greater than the model's prediction did not significantly help, either for workstations or for servers.
- For servers, a tighter range of T (1.5 standard deviations) was required in order to detect a significant proportion of problems. With $L = 1.0$, for the 10 servers, that cutoff yielded an average false positive rate of 1.8% (range 0–3%), and an average false negative rate of 20% (range 0–83%). For the three servers with the highest average load (1.2), $T = 1.0$ gave false positive rates averaging 5.7% and false negative rates averaging 32%.

Discussion

This analysis shows that the models provide excellent forecasts and provide useful problem identification, on the machines in our environment. The biggest weakness is that the average load is so low. However, this reflects the reality of our, and many others, environments. Servers are more difficult to analyze and will require more filtering by an expert or expert system, but typically they are few in number and file servers, at least, have limited access and are well-controlled. By contrast, users of public workstations are often naive, and frequently create problem (looping, etc.) processes which create poor response for the next user. Therefore, even though the average load on workstations is low, the likelihood of problems is significant.

5.2 SPA Expert System

The SPA expert system has undergone limited validation and tuning on a network of 102 workstations and servers. This was done using two methods: (1) the "trial by fire" approach, and (2) problem injection. Results of both of these methods are described in the following paragraphs.

Table 3: Time Series Model Analysis on Server Machines

		Minimum Forecast Error	False Positive Threshold			False Negative Threshold		
Host Name	Std. Dev.	(α, β, γ)	1.0	1.5	2.0	1.0	1.5	2.0
`sunset`	1.00	0.5, 0.2, 0.1	7	2	1	59	83	92
`cs`	0.74	0.9, 0.1, 0.2	1	0	0	14	62	96
`gr`	0.59	0.9, 0.1, 0.2	9	3	1	24	38	68
`asylum`	0.31	0.9, 0.1, 0.2	8	3	1	4	8	12
`jensen`	0.29	0.8, 0.1, 0.2	3	0	0	1	2	4
`vlsi`	0.26	0.9, 0.1, 0.2	4	2	1	3	5	8
`jaguar`	0.23	0.9, 0.1, 0.2	8	3	0	0	2	4
`peruvian`	0.20	0.9, 0.1, 0.2	9	3	1	0	0	1
`kayenta`	0.19	0.8, 0.1, 0.2	4	2	1	0	1	2
`ursa0`	0.06	0.9, 0.1, 0.2	1	0	0	0	0	0

The "Trial By Fire" Approach

In order to keep performance records, SPA records every time an instance of a problem is created. It also records the number of these problems that the system administrator classifies as valid and invalid. The following is a transcript of the output of the SPA `status` command. The `status` command is used to display the statistics that SPA maintains on problem detection:

```
SPA(H=3,int)> status
PROBLEM STATUS:
LOAD-HIGH:              7 valid /   14 found =   50.0 % success rate.
STATUS:                 8 valid /    9 found =   88.9 % success rate.
FILESERVER_DOWN:        2 valid /    2 found = 100.00 % success rate.
KERNEL_TABLE:           2 valid /    3 found =   66.7 % success rate.
MEM_SIZE:               2 valid /    4 found =   50.0 % success rate.
TIME_USED:              1 valid /    1 found = 100.00 % success rate.
ELAPSED_TIME:           0 valid /    1 found =    0.0 % success rate.
MEM_UTILIZATION:        3 valid /    5 found =   60.0 % success rate.
CPU_UTILIZATION:        3 valid /    4 found =   75.0 % success rate.
NFSD_PROCESSES:         0 valid /    0 found =    0.0 % success rate.
SCAN_RATE_PROBLEM:      5 valid /    6 found =   83.3 % success rate.
USER-COUNT-HIGH:        4 valid /    9 found =   44.4 % success rate.
WEDGED-PROCESS:         1 valid /    3 found =   33.3 % success rate.
ABANDONED-PROCESS:     11 valid /   15 found =   73.3 % success rate.
RENICED-PROCESS:        5 valid /    5 found = 100.00 % success rate.
ZOMBIE_PROCESS:         3 valid /    3 found = 100.00 % success rate.
SCAN_RATE_ALERT:        6 valid /    8 found =   75.0 % success rate.
PAGING_ALERT:           6 valid /    7 found =   85.7 % success rate.
OTHER:                  0 valid /    0 found =    0.0 % success rate.
TOTAL:                 69 valid /   99 found =   69.7 % success rate.

SPA(H=3,int)>
```

The biggest problem with using the "trial by fire" approach to validation of SPA is that it is a slow process. Bugs that are found have to be reported, fixed, and tested. Problems that SPA finds have to be investigated to determine their validity and then resolved. Some problem types occur frequently enough that it is possible to validate SPA's response when the problem occurs. Other problems occur so infrequently that it is impossible to thoroughly test SPA's response without some kind of problem injection.

In general, false positive results were too high for many problems such as memory utilization, for which we did not develop time series models. Either more tuning is necessary to better filter these, or, preferably, analyses could be run to develop time series models for these types of data.

Problem Injection

Problem injection was accomplished by introducing poorly-behaving programs into the environment, and auditing SPA to determine whether it correctly identified them. In general, it did very well for most problems which affected the load average, which is its screening device. `Wedged-process` and `load-high` were all detected at a high rate. `zombie-process`, `abandoned-process`, and `reniced-process` were detected whenever SPA's attention focused on a the host with a load average problem. `Elapsed time` worked well, when the rule was tuned to eliminate certain programs which users tend to keep running. Some other rules did not work as well. The main problem stems from SPA's primitive data acquisition facilities, that present too high an overhead to run frequently, on all hosts. This is an issue outside the domain of expert systems or the time series models, and is being addressed externally, through the SNMP Host MIB development.

6 Related Work

The work most similar to ours was done in the the Intrusion Detection Expert System[10] (IDES) and NIDX[2], computer security monitoring systems. They are based on an intrusion detection model described by Denning [3] for detecting security threats in computer systems. Denning's model is based on the hypothesis that a security violation can be detected by monitoring usage patterns of the system's users. A potential security violation would make itself evident as an abnormal pattern of usage, which is determined by comparing against predictions from time series models. The same idea is used by SPA to detect problems in hosts and processes.

TIMM/Tuner[16] is a computer system performance tuner developed in TIMM, a commercially available expert system shell. Its users are VMS system managers wishing to evaluate the system performance of a VAX computer system running VMS. The system attempts to isolate performance bottlenecks based on information supplied by the user in a question/answer session. The system recommends adjustments to system parameters, or if needed recommends hardware or software upgrades. Unlike SPA, TIMM/Tuner does not collect data in real-time. Although the problem domain is similar to that of SPA, it is a narrower domain because its expertise is limited to DEC computer systems running VMS.

Hitson described an expert system for monitoring and diagnosing problems in TCP/IP–based networks[5]. He concentrates on developing heuristics for diagnosing the ultimate problem, and does not use adaptive models to determine problem thresholds.

Network management systems such as HP's OpenView and Sun's SunNetmanager are an important related area. There are other commercial systems such as Tivoli's, Cabletron's Spectrum, and the forthcoming DME from OSF. Most of these systems are simply frameworks for other (sometimes non-existent) tools, which do the actual work. In the absence of automated tools, the management systems provide a framework into which the system manager inserts simple, limited relations. As far as we can determine, none of these systems even dynamically determines threshold values, although most can deal with proportional measures. They lack the capability of adapting to changes in the data. Typically, the network administrator spends a considerable amount of time determining appropriate threshold values for the data monitored in the network. If the usage of the network changes significantly, threshold values may become invalid (causing either a flood of alarms or a complete absence of alarms). Our time series models address this problem by adapting to changes in system usage.

Because current commercial computer and network management tools have limited abilities to combine criteria in complex ways, they require extensive per-host configuration to detect particular problems. SPA provides much more power and flexibility with its support of an SQL subset, its ability to be extended at run time, and its powerful and extensible knowledge base facts and rules.

Standardized SNMP Host MIBs that enable better, faster, and cheaper access to measurements of host state are an important step in the development of better diagnostic and management tools.

7 Future Work

SPA is a useful prototype to validate our time series model and the expert system itself. However, as a large Lisp system, it is undesirably costly in processor cycles and memory use. Portions of it could be re-implemented in C++. The use of standard (and costly) Unix utilities to gather raw data could be replaced by a custom daemon. (When SNMP host MIB's are widely available, those can be used.) With these efficiency improvements, SPA could prove of significant practical use.

There are many additional problem areas SPA might diagnose, such as the network itself, disk utilization, and system security. One of the problems in monitoring all the hosts in a large network is the volume of problems, alerts, and messages that gets generated. A more effective means of filtering this information could be provided, or more aggressive use of time series models could be used.

Additional extensibility of the SPA system could be provided. The `include` command provides a primitive form of extensibility by allowing a means by which the system administrator can define new problem types. However, frequently, it is useful for the system administrator to define procedural actions to take when specific conditions occur. By allowing the system administrator to specify an arbitrary set of actions in the `include` command, powerful extensibility can be provided.

8 Conclusion

A time series model is an effective, practical, easy to implement technique for determining problem threshold values in computer systems. It alleviates one of the most significant weaknesses of current computer and network management systems: the manual determination of a fixed threshold value. The effectiveness of the time series model is being demonstrated by its use in the SPA system administration expert system. The sophistication and power of an expert system is appropriate, and we suspect required, for effective system management in today's large computer networks.

Acknowledgements

We are grateful to Sean O'Neill for analyzing the times series models. Our thanks for reviewing earlier drafts of this paper go to Robert Kessler, Mike Hibler, and Sean.

References

[1] American National Standards Institute, New York, NY. *X3.135-1989: Database Language — SQL with Integrity Enhancement*, 1989.

[2] David S. Bauer and Michael E. Koblentz. NIDX – a real-time intrusion detection expert system. In *Proceedings of the Summer 1988 USENIX Conference*, pages 261–273, 1988.

[3] Dorothy E. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, February 1987.

[4] Domenico Ferrari, Giuseppe Serazzi, and Alessandro Zeigner. *Measurement and Tuning of Computer Systems.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[5] Bruce L. Hitson. Knowledge-based monitoring and control: an approach to understanding the behavior of TCP/IP network protocols. *Proc. of the SIGCOMM '88 Symposium on Communication Architectures and Protocols*, 18(4):210–221, August 1988.

[6] Peter James Hoogenboom. Semantic definition of a subset of the structured query language (SQL). Technical Report UUCS-91-026, University of Utah, December 1991.

[7] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

[8] Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison-Wesley Publishing Company, Reading, MA, 1989.

[9] Colin D. Lewis. *Industrial and Business Forecasting Methods.* Butterworth Scientific, London, 1982.

[10] Teresa F. Lunt and R. Jagannathan. A prototype real-time intrusion-detection expert system. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 59–66. IEEE Computer Society Press, 1988.

[11] Douglas C. Montgomery, Lynwood A. Johnson, and John S. Gardiner. *Forecasting and Time Series Analysis.* McGraw-Hill, Inc., New York, NY, 2nd edition, 1990.

[12] Eric George Muehle. FROBS: a merger of two knowledge representation paradigms. Master's thesis, University of Utah, 1987.

[13] J. H. Saltzer and J. W. Gintell. The instrumentation of MULTICS. *Communications of the ACM*, 13(8):495–500, 1970.

[14] Robert Sedgewick. *Algorithms.* Addison-Wesley Publishing Company, Reading, MA, 2nd edition, 1988.

[15] Brian Sturgill. System administration tools. Master's thesis, University of Kentucky, 1989.

[16] Donald A. Waterman. *A Guide to Expert Systems.* Addison-Wesley Publishing Company, Reading, MA, 1986.

[17] Peter R. Winters. Forecasting sales by exponentially weighted moving averages. *Management Science*, 6(3):324–342, April 1960.

Author Information

Peter Hoogenboom recently received his Masters degree at the University of Utah. The SPA expert system was the subject of his Masters thesis. In his eight years of experience in the industry, Peter has worked in a variety of capacities, including the design and analysis of real-time simulation systems, system administration, and expert systems for UNIX system administration. His current projects include porting GNU tools to the HP PA and the development of the OMOS Object File Editor (OFE). Since graduating with his Masters degree, Peter has become a full-time staff member in the Center for Software Science.

Jay Lepreau is Assistant Director of the Center for Software Science, a research group within Utah's Computer Science Department which works in many aspects of systems software. He has worked with Unix since 1979, and has served as co-chair of the 1984 USENIX conference and on numerous other USENIX program committees. His group has made significant contributions to the BSD and GNU software distributions. His current research interests include dynamic software system structuring for performance and flexibility, with operating system, language, linking, and runtime components.

The author's addresses are: Center for Software Science, Department of Computer Science, University of Utah, 84112. They can be reached electronically at {hoogen,lepreau}@cs.utah.edu.