System Performance Advisor User Guide[1]

Peter J. Hoogenboom

UUCS-91-025

Department of Computer Science
University of Utah
Salt Lake City, UT 84112, USA

December 9, 1991

Abstract

The usage of the System Performance Advisor (SPA) expert system is described. Documentation of SPA system commands, system variables, diagnostic rules is given. Information on how to run the SPA system is discussed. In addition, an overview of how SPA searches for problems is supplied. The purpose of SPA is to assist a system administrator in system performance management. Generally, system performance management can be described as those activities performed by a system administrator to ensure a computer system is providing as much of its capacity for the use of its users. SPA assists in performing these duties by attempting to locate the source of performance problems and then notifying the system administrator. SPA makes diagnoses about hosts and processes by monitoring current levels of activity and maintaining a history of these activity levels. By comparing historic knowledge of behavior to dynamic (current) knowledge, SPA can detect when a system is experiencing performance problems.

[1]This work was supported in part by the Hewlett-Packard Corporation

Contents

1 Introduction to SPA 2
1.1 Notational Conventions . . . 2
1.2 Getting Started . . . 2
1.2.1 Files Accessed in a SPA Session . . . 3
1.3 Interacting With SPA . . . 3
1.3.1 The SPA Prompt . . . 4
1.3.2 Names of Knowledge Base Objects . . . 4

2 How SPA Looks For Problems 6
2.1 A Model of the System Performance Management Process . . . 6
2.1.1 Data Collection Phase . . . 6
2.1.2 Data Analysis Phase . . . 7
2.1.3 Problem Therapy Phase . . . 7
2.2 The SPA Implementation of the Model . . . 7

3 SPA Problem Definitions 12

4 SPA Rule Summary 14

5 SPA Commands 28

6 Maintaining SPA 35
6.1 Customizing SPA With the Initialization File: `.sparc` . . . 35
6.1.1 Required Information . . . 35
6.1.2 Optional Information For Customizing SPA . . . 35
6.2 How to Add a Monitored Statistic to SPA . . . 36

7 Customizing SPA 38
7.1 Defining Host Profiles . . . 40
7.2 Defining Program Profiles . . . 40
7.3 Other parameters . . . 41

1 Introduction to SPA

This document provides an introduction to the use of the **System Performance Advisor** (SPA) expert system. SPA is used by UNIX[1] system administrators to assist them in maintaining maximum processor performance for all processors in a network.

This document is intended primarily as a reference manual for users who are already familiar with the terms and concepts of the SPA expert system.

Seven elements of SPA are discussed here. They are:

- Introduction to SPA.
- How SPA searches for problems.
- Problems that SPA looks for.
- SPA rule summary.
- SPA command summary.
- Maintaining SPA.
- Customizing SPA.

1.1 Notational Conventions

In this user guide, actual code (that is, names of functions, variables, commands, etc.) that appears on the computer screen or is typed by the user on a keyboard is shown in a typewriter font. For example, `this is the typewriter font`. Names in this user guide that represent names of functions, variables, or commands are shown in *italics.*

In Section 5, a description of SPA commands is given. In the descriptions of the syntax of commands, the items enclosed in square brackets (`[ ... ]`) indicate optional items. The vertical bar ('|') is used to separate alternative arguments.

1.2 Getting Started

Currently, the SPA executable resides on the host `jaguar.cs.utah.edu`. The recommended configuration to run SPA is to have a sub-directory called `SPA` and a SPA initialization file (`.sparc`) in your home directory. A copy of the SPA initialization file is available in `/n/jaguar/u/hoogen`. Within the SPA sub-directory, there should be a soft link (see the `man` page for `ln(1)`) installed to access the SPA executable. Assuming you haven't created the `SPA` sub-directory yet, the following steps should be performed prior to running SPA:

1. `cd`
2. `cp /n/jaguar/u/hoogen/.sparc .`
3. `mkdir SPA`
4. `ln -s /n/jaguar/u/hoogen/SPA/spa spa`

Once these steps have been performed, SPA can be run with the following commands:

[1] UNIX is a trademark of AT&T Laboratories.

```
cd SPA
spa
```

The SPA system is written in Common Lisp: if an error occurs, rather than exiting to the UNIX shell, SPA exits to the top-level loop of Common Lisp. Unless a serious error has been encountered, you may re-enter SPA with the following command:

```
:a
```

Normally, this should take you back to a SPA prompt that looks like "`SPA(H=3,int)>` ". If not, try the following sequence of commands:

```
:a
(spa)
```

1.2.1 Files Accessed in a SPA Session

There are several files that SPA accesses in the course of a session. These files are listed and described here.

`.sparc`: The SPA startup file (if it exists) is read during SPA initialization. Parts of the contents of this file are required for SPA to operate correctly. Thus, it is recommended that the user have a `.sparc` file in his/her home directory. The contents of the `.sparc` file are described fully in Section 6.

`spa.stats`: SPA maintains a statistics file profiling the performance of SPA. During initialization, SPA will read the statistics regarding previous sessions of SPA (contained in `spa.stats` in the directory from which SPA was started). Upon termination of SPA, this file is updated.

Note that because SPA expects this file to be in the directory from which SPA was started, it is recommended that the user start each SPA session from the same directory.

`cmd-history-`*MM-DD-YYYY*: Each SPA session is recorded in a history file whose name is of the form:

`cmd-history-`*MM-DD-YYYY*

where *MM* is the month ($01 \leq MM \leq 12$), *DD* is the date ($01 \leq DD \leq 31$), and *YYYY* is the year. This file is written in the directory from which you started SPA.

`/n/jaguar/u/hoogen/HOSTHIST/HOSTS/*`: The files in this directory are read during a SPA session to retrieve the load average histories of hosts.

`/n/jaguar/u/hoogen/HOSTHIST/PROCESSES/*`: The files in this directory are read during a SPA session to retrieve the histories of programs that have been executed.

1.3 Interacting With SPA

Once SPA has been started, how (and when) does one specify commands in order to get SPA to assist the system administrator? Before detailing individual commands (individual commands are discussed in Section 5), some basic conventions adhered to by SPA should be explained.

1.3.1 The SPA Prompt

Whenever the SPA prompt is displayed, SPA is asking the user for some kind of input. A command can be continued from one line to the next by ending the line with a single backslash ("\").

If there is no prompt, SPA is performing some action such as data collection or analysis. When SPA is performing some action, do not attempt to give SPA some input because it will be ignored. While SPA is performing some kind of action, it will periodically display a status message to indicate what it is doing.

Not only does the prompt indicate when the user can interact with SPA, it also provides information about what modes are active. Normally, the prompt looks like this[2]:

`SPA(H=`*N*`,`*int-flag*`,`*debug-flag*`)>`

The meanings of these mode indications are given in Table 1.

Table 1: User Prompt Modes Display

Mode	Description
`H=`*N*	
	Current setting of **`history`**. **`history`** defines how much information is logged in the command history file. *N* is the history level: 0—history is disabled 1—command history is maintained 2—command and action history is maintained 3—history level 2 plus diagnostic output is provided
int-flag	
`int`	Interactive mode is enabled.
`int=`*N*	Interactive mode is enabled. *N* iterations through the performance management process will be performed.
debug-flag	
`dbg`	Debug mode. Additional information regarding inferences and data collection will be displayed.

1.3.2 Names of Knowledge Base Objects

Knowledge base objects are data structures that SPA maintains representing objects with which SPA is concerned: hosts, processes, host histories, process histories, statistics, time series models, and so on.

Hosts Hosts are named according to the names that appear in the output of a `ruptime`. Thus, all the current data in the knowledge base about the host `asylum` can be gotten with the command:

```
show asylum
```

[2]The prompt can be changed with the SPA `set` command

Processes Processes are named by their process identification numbers (PID). But, since SPA collects process information on more than one host, uniqueness of PIDs is not guaranteed. SPA enforces uniqueness of process names by appending to the PID "`@hostname`" where `hostname` is the host where the PID is executing. These two items together become the full name of the process. The name can be shortened, however, to just the PID. When only the PID is specified, the current host is assumed[3]. Thus, if the current host is `shafer`,

```
show 1234@shafer ≡ show 1234
```

Other Knowledge Base Objects There are many other objects in the knowledge base besides hosts and processes. Most of them are associated with a host or process, however. Sometimes, the user would like to examine one or more of these other objects. Or, for a given host or process, he/she is only interested in one value. The SPA `show` command allows the user to specify these kinds of queries with a syntax that is similar to how individual fields of a `struct` are accessed in the C programming language.

For example, if the user only wants to display the user count on the host `cs`, the following command will do that:

```
show cs.load15
```

The history data that SPA maintains on a host is a large data structure with many fields: each host has this data structure in a slot called `history_frob`. The history data can be displayed with the `show history` command. For example,

```
show history peruvian
```

shows the history data for the host `peruvian`. Alternatively, the user could enter:

```
show peruvian.history_frob
```

to show the same data.

As another example, suppose the user wants to display the statistics regarding the memory utilization of a process whose PID is 1389 executing on the host `cs`. The statistics are historical data, so the data that is needed is in the historical data stored in the process's `history_frob` slot. Not all the historical information is needed: only the `mem` slot. Thus, the following command will display exactly what is wanted:

```
show 1389@cs.history_frob.mem
```

[3]The current host can be changed with the `chost` command.

2 How SPA Looks For Problems

2.1 A Model of the System Performance Management Process

The process of managing a system's performance is an iterative one. As shown in Figure 1, it is composed of a *diagnosis* phase and a *therapy* phase. These phases are repeated until there are no known performance problems in the system. The first step in the diagnosis phase consists of making measurements of the system. The second step in diagnosis is to analyze the data that has been collected. During analysis, the performance problems are recognized and categorized. Once the analysis is done, and the results show that some improvements in performance are possible, a therapy phase is begun. During this phase, one or more adjustments to the system are made in hopes of alleviating the problem, then the system is measured again. If the results of the analysis are inconclusive, more data or different data can be collected, or if it can be shown that inconclusive results imply that no further performance improvements are possible, the management process can terminate.

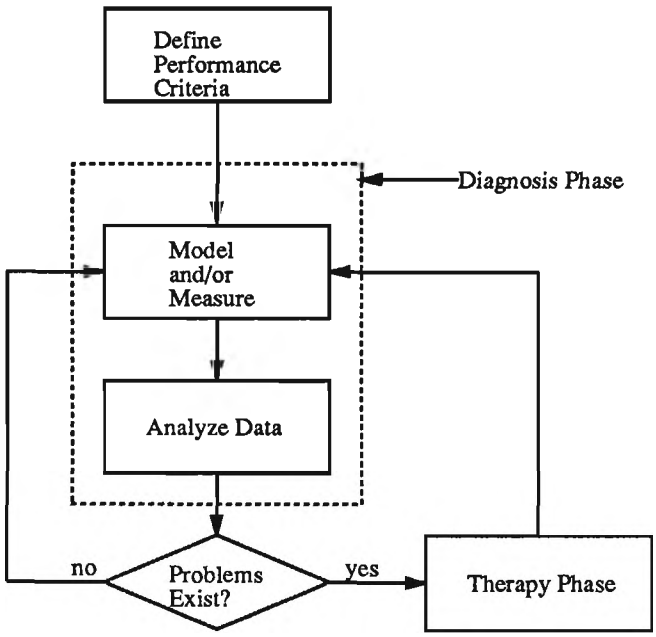

Figure 1: The Process of System Performance Management

2.1.1 Data Collection Phase

SPA collects data concerning computer systems (called *hosts*) and the tasks running on those hosts (called *processes*).

Hosts Hosts are computer systems and all their supporting software and hardware. Historical information about various parameters of hosts (e.g. load average, users, amount of memory in use) is maintained by SPA to allow SPA to determine whether a problem exists on that host.

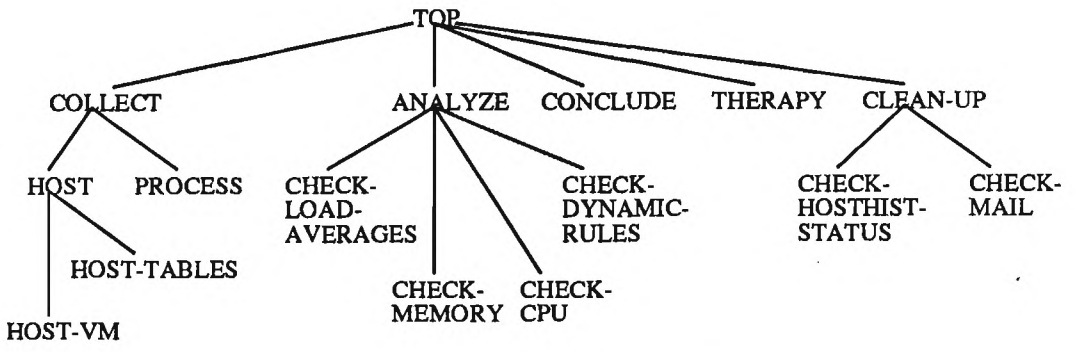

Figure 2: SPA Sub-plan Search Tree

By comparing the current values of these parameters to what has been experienced in the past, *abnormal* behavior is detected. Table 2 is a description of the information that SPA maintains about hosts.

Processes A process is a running program on a host. SPA maintains histories of several characteristics of processes in order to know what is normal or abnormal behavior for a process. By comparing the current values of performance criteria to what has been experienced in the past, *abnormal* behavior is detected. Table 3 is a description of the information that SPA maintains about processes.

2.1.2 Data Analysis Phase

Data analysis is accomplished in SPA by using forward-chaining rules to detect deviations of behavior from the expected profiles built from historical data. These rules will build instances of problems that can be examined during the problem therapy phase.

2.1.3 Problem Therapy Phase

Usually, problem therapy is accomplished by the system administrator. As its name indicates, SPA acts as an *advisor* to assist the system administrator in locating the source of the problem. Once problems are located, the system administrator can consult SPA to obtain an explanation of the problem, suggested plans to solve the problem, and suggested utilities that can be executed to obtain additional information.

2.2 The SPA Implementation of the Model

Each phase or *plan* is divided into zero or more sub-phases or *sub-plans*. This allows SPA to look for certain kinds of problems or perform certain kinds of tasks at one particular time within the phase. This avoids an explosion in the number of possible tasks to perform at a given time. These sub-plans are sequenced so that SPA considers one sub-plan at any one time. The sequencing of sub-plans that SPA performs is equivalent to a pre-order traversal of the sub-plan tree shown in Figure 2. The following paragraphs describe the sub-plans that are currently implemented in SPA.

Table 2: Host Data Items

Slot Name		Description
NAME	—	name of the host (e.g. **cs**)
STATUS	—	**up** or **down**
USER_COUNT	—	number of users on the host
LOAD1	—	1-minute load average
LOAD5	—	5-minute load average
LOAD15	—	15-minute load average
AVM	—	active virtual pages
FRE	—	amount of memory in the free list
PGIN	—	pages paged in
PGOUT	—	pages paged out
IN	—	(non clock) device interrupts per second
SY	—	system calls per second
CS	—	cpu context switch rate (switches/sec)
SR	—	paging algorithm scan rate
DE	—	expected short-term memory shortfall
FILES_USED	—	kernel file table entries used
TOTAL_FILES	—	total kernel file table entries
INODES_USED	—	kernel inode table entries used
TOTAL_INODES	—	total kernel inode table entries
PROCESSES_USED	—	kernel process table entries used
TOTAL_PROCESSES	—	total process table entries
TEXTS_USED	—	kernel program text table entries used
TOTAL_TEXTS	—	total program text table entries
SWAP_IN_USE	—	kernel swap slots in use
TOTAL_SWAP	—	total swap slots available
SM_SEGMENTS_USED	—	shared memory segments used
TOTAL_SM_SEGMENTS	—	total shared memory segments
SEMAPHORES_USED	—	semaphores in use
TOTAL_SEMAPHORES	—	total number of semaphores available
MSG_QUEUES_USED	—	message queues in use
TOTAL_MSG_QUEUES	—	total message queues available
BOOT_TIME	—	last reboot time
REAL_MEM	—	total amount of real memory
AVAIL_MEM	—	total amount of available memory
HISTORY_FROB	—	the historical profile for the host

Table 3: Process Data Items

Slot Name		Description
NAME	—	the program name (i.e. argv[0] used for accounting)
USER	—	the user-id of the owner of this process
UID	—	the integer form of the user-id
PID	—	process id. This value can be used to refer to this process. For example, **show 12345 12121** will display the current values of the variables listed in this table for the processes with pid values of 12345 and 12121.
PPID	—	the parent process of this process
PRIORITY	—	the priority of the process (PZERO = 0)
NICE	—	nice factor (NZERO = 0)
MEM	—	percentage of real memory used by this process.
CPU	—	cpu utilization of the process; this is a decaying average over up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all CPU fields to exceed 100%.
SIZE	—	virtual size of the process (in 1024 byte units)
RSS	—	real memory (resident set) size of the process (in 1024 byte units)
WCHAN	—	event on which process is waiting (an address in the system). A symbol is chosen that classifies the address.
TTY	—	controlling tty of this process
STATUS	—	status flags for this process.
TIME_USED	—	amount of CPU time consumed by this process
HOST_NAME	—	the name of the host (e.g. jensen.utah.edu) where this process is running
CHILD_PID_LIST	—	child processes of this process

TOP: At the top-level, SPA performs operations of data collection, analysis, derivation of conclusions, therapy, and internal maintenance repetitively in search of performance problems.

TOP sub-plans: collect analyze conclude therapy clean-up

COLLECT: SPA collects current values of a variety of host and process performance criteria in search of performance problems.

COLLECT sub-plans: host process

ANALYZE: SPA checks the most recent data that was collected against historical profiles of the data. Any unexpected values are noted for later reference.

ANALYZE sub-plans: check-load-averages check-memory check-cpu check-dynamic-rules

CHECK-LOAD-AVERAGES: The CHECK-LOAD-AVERAGES phase involves examining the current load average measurements and comparing them to a profile of previous measurements. The previous profile is maintained as seasonally (daily) adjusted time series.

CHECK-LOAD-AVERAGES sub-plans: None

CHECK-MEMORY: The CHECK-MEMORY phase involves examining the current memory usage of all processes currently executing on a host. These measurements are compared to a profile of previous measurements. The previous profile is maintained as seasonally (daily) adjusted time series.

CHECK-MEMORY sub-plans: None

CHECK-CPU: The CHECK-CPU phase involves examining the current CPU usage of all processes currently executing on a host. These measurements are compared to a profile of previous measurements. The previous profile is maintained as seasonally (daily) adjusted time series.

CHECK-CPU sub-plans: None

CHECK-DYNAMIC-RULES: The CHECK-DYNAMIC-RULES phase involves checking for conditions that the user has identified as additional problems (via the `include` command).

CHECK-DYNAMIC-RULES sub-plans: None

CONCLUDE: The CONCLUDE phase involves deriving conclusions about possible problems that have been found.

CONCLUDE sub-plans: None

THERAPY: Therapy involves determining one or more appropriate actions that can be taken in order to resolve a performance problem.

THERAPY sub-plans: None

HOST: Data collection regarding hosts involves collection of performance criteria from utilities such as **ruptime**, **vmstat**, **iostat**, and **pstat**.

HOST sub-plans: host-vm host-tables

HOST-VM: Data collection regarding virtual memory on a host involves collection of performance criteria from the **vmstat** utility.

HOST-VM sub-plans: None

HOST-MEM: Data collection regarding the memory configuration of a host involves parsing **/usr/adm/messages** looking for the **real** and **avail** memory entries. The most recent entry is found and used.

HOST-MEM sub-plans: None

REMOTE-HOST-MEM: Data collection regarding the memory configuration of a host involves parsing **/usr/adm/messages** looking for the **real** and **avail** memory entries. The most recent entry is found and used.

REMOTE-HOST-MEM sub-plans: None

HOST-TABLES: Data collection regarding a host's system tables involves collection of information about how full certain kernel tables have become. This data is collected via the **pstat** utility. Information on the current fullness of the open file table, inode table, process table, swap space, shared memory segments, semaphores, and message queues is maintained.

HOST-TABLES sub-plans: None

PROCESS: Data collection regarding processes involves collection of performance criteria from utilities such as **ps**.

PROCESS sub-plans: None

CLEAN-UP: SPA performs maintenance of internal data structures in order to ensure optimal performance of itself.

CLEAN-UP sub-plans: check-hosthist-status

CHECK-HOSTHIST-STATUS: HOSTHIST is the program that collects historical data on the load averages of hosts. SPA makes sure that it is running during this phase.

CHECK-HOSTHIST-STATUS sub-plans: None

3 SPA Problem Definitions

Each condition that could be causing a system performance problem is called a *problem*. A condition that is less serious or is merely an informational item that the SPA user might be interested in knowing about is called an *alert*. In terms of object-oriented programming, the class of problems is a sub-class of the class of alerts. You can think of a problem as an alert with a raised priority or severity rating. Each alert or problem type that SPA attempts to locate is assigned a type name. When the SPA system displays the information about a particular instance of a problem or alert, this name is displayed. A diagram of the hierarchy of alerts and problems that SPA attempts to locate is shown in Figure 3. In addition, the following paragraphs give a brief description of the alert or problem.

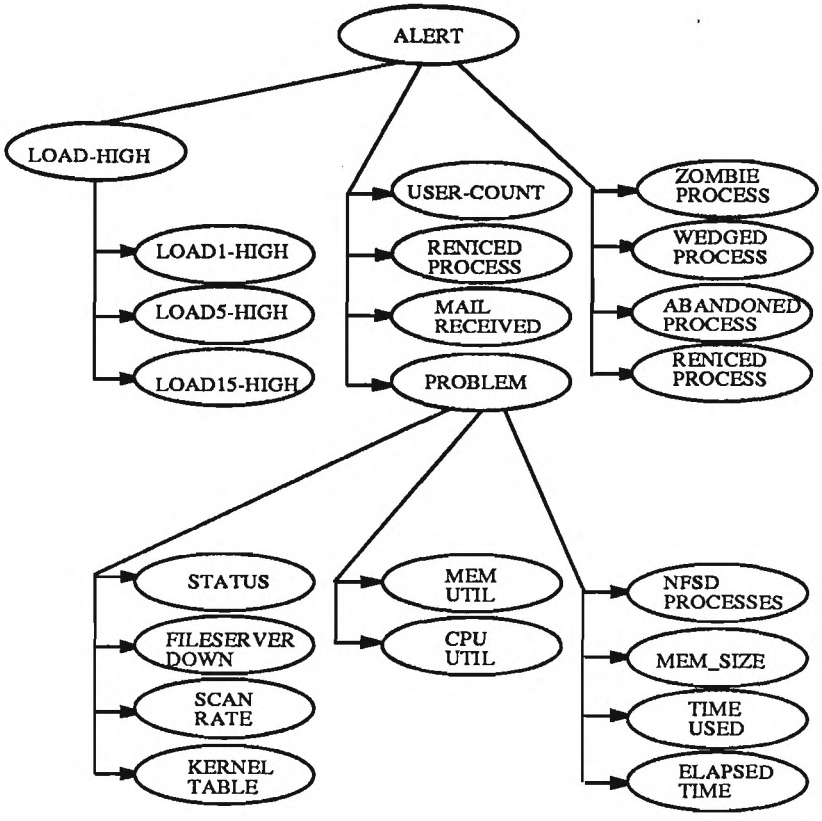

Figure 3: SPA Alert and Problem Classes

ALERT Sometimes, an alert is a condition that exists that is either temporary (such as `mail_received` alert) and is only shown once to the user, or it is a condition that SPA is currently investigating (such as the family of `load_high` alerts) and further information will be provided after SPA has collected additional data.

LOAD-HIGH One of the load average measurements is higher than expected.

LOAD1-HIGH The current 1-minute load average is higher than expected. To determine this, SPA compares the current 1-minute load average with a forecast of the 1-minute load average. If the two differ significantly, this alert is created. The forecast is made based on the data stored in a time series model. This model is created from a history of all 1-minute load averages on the host.

LOAD5-HIGH The current 5-minute load average is higher than expected. To determine this, SPA compares the current 5-minute load average with a forecast of the 5-minute load average. If the two differ significantly, this alert is created. The forecast is made based on the data stored in a time series model. This model is created from a history of all 5-minute load averages on the host.

LOAD15-HIGH The current 15-minute load average is higher than expected. To determine this, SPA compares the current 15-minute load average with a forecast of the 15-minute load average. If the two differ significantly, this alert is created. The forecast is made based on the data stored in a time series model. This model is created from a history of all 15-minute load averages on the host.

USER-COUNT-HIGH This host has more users logged in than expected. To determine this, SPA compares the current 1-minute load average with a forecast of the 1-minute load average. If the two differ significantly, this alert is created. The forecast is made based on the data stored in a time series model. This model is created from a history of all 1-minute load averages on the host.

Short of rebooting the host, there usually isn't anything that can be done to resolve this condition.

WEDGED-PROCESS The load average is fixed at X.00. This could indicate a wedged process.

ABANDONED-PROCESS This process does not have a parent process owned by the user. This could indicate a process started and then forgotten when the user logged off. It could also be a background job started by the user.

RENICED-PROCESS This process has been reniced to a higher priority. Some processes are normally executed at a higher priority. However, these processes are system processes (usually owned by `root`).

MAIL_RECEIVED Mail has been received regarding a problem.

ZOMBIE_PROCESS This process is in a final exit state.

SCAN_RATE_ALERT This host is loaded heavily enough to cause paging. The number of pages per second scanned by the clock algorithm is non-zero.

PAGING_ALERT This host is loaded heavily enough to cause paging. The amount of memory in use is greater than the amount of real memory on the system.

STATUS The host is down. Or the host has not broadcast its **rwhod** data.

FILESERVER_DOWN The host (a fileserver) is down. Or the host has not broadcast its **rwhod** data.

KERNEL_TABLE This host has one or more of its kernel tables approaching 100% full.

MEM_SIZE This process is using more memory than expected.

TIME_USED This process has accumulated more CPU time than expected.

ELAPSED_TIME This process has been executing for a longer time than expected.

MEM_UTILIZATION This process is utilizing more memory than expected.

CPU_UTILIZATION This process is utilizing more of the CPU than expected.

NFSD_PROCESSES There are a number of 'nfsd' processes on a host. All of them are near the top of the list in terms of CPU utilization. Also, the load average is high on this host (at or close to N where N is the number of 'nfsd' processes executing on this host).

One cause of this kind of condition is a 'find' process that is accessing filesystems over NFS.

SCAN_RATE_PROBLEM This host is loaded heavily enough to cause paging. The paging rate is higher than expected. This condition could warrant investigation to make sure the system is operating normally.

4 SPA Rule Summary

1. RULE-COLLECT-TABLE-DATA

```
IF
    SPA is currently collecting HOST-TABLE data on hosts
  AND
    there exists a host H for which we have
    diagnosed a problem
  AND
    H is up
  AND
    H is the local host
THEN
  collect HOST-TABLE statistics on the local host.
```

2. RULE-COLLECT-TABLE-DATA-REMOTE

```
IF
    SPA is currently collecting HOST-TABLE data on hosts
  AND
    there exists a host H for which we have
```

```
    diagnosed a problem
  AND
    H is up
  AND
    H is not the local host
  AND
    H is one for which SPA collects HOST-TABLE statistics
THEN
  collect HOST-TABLE statistics on that host.
```

3. RULE-FILE-TABLE-FULL

```
IF
    SPA is currently in the analysis phase
  AND
    there exists a host H
  AND
    H is not already involved in a problem of this type
  AND
    the file table for H is 90% full or more
THEN
  create a problem of type KERNEL_TABLE
```

4. RULE-INODE-TABLE-FULL

```
IF
    SPA is currently in the analysis phase
  AND
    there exists a host H
  AND
    H is not already involved in a problem of this type
  AND
    the inode table for H is 90% full or more
THEN
  create a problem of type KERNEL_TABLE
```

5. RULE-PROCESS-TABLE-FULL

```
IF
    SPA is currently in the analysis phase
  AND
    there exists a host H
  AND
    H is not already involved in a problem of this type
  AND
    the process table for H is 90% full or more
THEN
  create a problem of type KERNEL_TABLE
```

6. RULE-TEXTS-TABLE-FULL

```
IF
    SPA is currently in the analysis phase
  AND
    there exists a host H
  AND
    H is not already involved in a problem of this type
  AND
    the texts table for H is 90% full or more
THEN
  create a problem of type KERNEL_TABLE
```

7. RULE-SM-SEGMENT-TABLE-FULL

```
IF
    SPA is currently in the analysis phase
  AND
    there exists a host H
  AND
    H is not already involved in a problem of this type
  AND
    the shared memory segment table for H is 90% full or more
THEN
  create a problem of type KERNEL_TABLE
```

8. RULE-SEMAPHORE-TABLE-FULL

```
IF
    SPA is currently in the analysis phase
  AND
    there exists a host H
  AND
    H is not already involved in a problem of this type
  AND
    the semaphore table for H is 90% full or more
THEN
  create a problem of type KERNEL_TABLE
```

9. RULE-MSG-QUEUE-TABLE-FULL

```
IF
    SPA is currently in the analysis phase
  AND
    there exists a host H
  AND
    H is not already involved in a problem of this type
  AND
    the message queue table for H is 90% full or more
THEN
  create a problem of type KERNEL_TABLE
```

10. RULE-HOST-PAGING-ALERT

```
IF
    SPA is currently in the analysis phase
  AND
    there exists a host H
  AND
    H is not already involved in a problem of this type
  AND
    the number of physical memory pages in use is ?avm
  AND
    the amount of real, available physical memory is ?avail_mem
  AND
    ?avm >= (?avail_mem / 1024)
    (the number of pages in use is at or above the amount of
    real, available memory)
THEN
  the system is paging, so create an alert of type PAGING-ALERT
```

11. RULE-HOST-SCAN-RATE-ALERT

```
IF
    SPA is currently in the analysis phase
  AND
    there exists a host H
  AND
    H is not already involved in a problem of this type
  AND
    the virtual memory clock algorithm scan rate is non-zero
THEN
  create an alert of type SCAN_RATE-ALERT
```

12. RULE-HOST-SCAN-RATE-PROBLEM

```
IF
    SPA is currently in the analysis phase
  AND
    there exists a host H
  AND
    there is a scan rate alert in effect for host H
  AND
    the current virtual memory clock algorithm scan rate is ?SR
  AND
    ?SR is higher than expected (The expected value is
    determined from the average of scan rate readings on this
    host.)
THEN
  create a problem of type SCAN_RATE-PROBLEM
```

13. RULE-COLLECT-HOST-MEM

```
IF
    SPA is currently looking for the host memory configuration
  AND
    there exists a host H
  AND
    H is the local host
  AND
    SPA has not collected the memory configuration for H yet
THEN
  perform the host memory configuration data collection process
```

14. RULE-COLLECT-REMOTE-HOST-MEM

```
IF
    SPA is currently looking for the host memory configuration
  AND
    there exists a host H
  AND
    SPA has diagnosed a problem or alert on H
  AND
    H is up
  AND
    H is not the local host
  AND
    H is a host for which SPA collects the host memory configuration
  AND
    SPA has not collected the memory configuration for H yet
THEN
  perform the remote host memory configuration data collection process
```

15. RULE-DELETE-HOST-INSTANCES

```
IF
    SPA is currently deleting instances of hosts
  AND
    there exists a host H
  AND
    SPA has diagnosed a problem or alert involving H
  AND
    H is the only instance of a host or process
    involved in this problem.
THEN
  delete the instance of the problem or alert.
```

16. RULE-DIAG-HOST-LA1

```
IF
     SPA is currently analyzing data
  AND
```

```
     the load1 on the host H is ?LOAD1
  AND
     H is not already involved in a load1 problem
  AND
     ?LOAD1 is higher than expected
     (equivalent to checking to see if the current
      forecast error is negative and larger than
      *std-dev-limit* standard deviations from the
      average forecast error)
THEN
  create an instance of a load1 problem.
```

17. RULE-DIAG-HOST-LA5

```
IF
     SPA is currently analyzing data
  AND
     the load5 on the host H is ?curr-load
  AND
     H is not already involved in a load5 problem
  AND
     ?curr-load is higher than expected
     (equivalent to checking to see if the current
      forecast error is negative and larger than
      *std-dev-limit* standard deviations from the
      average forecast error)
THEN
  create an instance of a load5 problem.
```

18. RULE-DIAG-HOST-LA15

```
IF
     SPA is currently analyzing data
  AND
     the load15 on the host H is ?curr-load
  AND
     H is not already involved in a load15 problem
  AND
     ?curr-load is higher than expected
     (equivalent to checking to see if the current
      forecast error is negative and larger than
      *std-dev-limit* standard deviations from the
      average forecast error)
THEN
  create an instance of a load15 problem.
```

19. RULE-DIAG-HOST-USER-COUNT

```
IF
     SPA is currently checking load averages
  AND
     the current number of users on host H is U
```

```
  AND
     H is not already involved in a user-count problem
  AND
     U is higher than expected
     (equivalent to checking to see if the current
      forecast error is negative and larger than
      *std-dev-limit* standard deviations from the
      average forecast error)
THEN
  create an instance of a user-count problem.
```

20. RULE-DIAG-HOST-DOWN

```
IF
     SPA is currently checking load averages
  AND
     the current status host H is 'down'
  AND
     H is not already involved in a status problem
THEN
  create an instance of a status problem.
```

21. RULE-DIAG-FILESERVER-DOWN

```
IF
     SPA is currently checking load averages
  AND
     the current status host H is 'down'
  AND
     H is a FILESERVER
  AND
     H is not already involved in a status problem
THEN
  create an instance of a status problem.
```

22. RULE-LOAD-AVE-PEGGED-AT-X.00

```
IF
    SPA is currently checking load averages
  AND
    Host H is a workstation
  AND
    H is up
  AND
    load1 of H is L1
  AND
    load5 of H is L5
  AND
    load15 of H is L15
  AND
    L1 = L5 = L15
```

```
  AND
    L1 > 0.00
  AND
    the fractional part of L1 is .00 (L1 is expressible as an integer)
  AND
    H is not involved in a wedged process problem already
THEN
  create an instance of a wedged process problem.
```

23. RULE-DIAG-HOST-LA-ABS-LIMIT

```
IF
    SPA is currently checking load averages
  AND
    there exists a host H
  AND
    H is not already involved in a load average problem
  AND
    (
        the load1 reading of H is above
        the absolute allowable maximum
      OR
        the load5 reading of H is above
        the absolute allowable maximum
      OR
        the load15 reading of H is above
        the absolute allowable maximum
    )
THEN
  create an instance of loadX-high problem (X=1, 5, or 15)
```

24. RULE-DELETE-EXPIRED

```
IF
    SPA is currently doing clean-up
  AND
      there exists a problem FROB with a time limit
  AND
      the time limit has expired
THEN
  delete the problem FROB
```

25. RULE-DELETE-EXPIRED-DYNAMIC-RULE

```
IF
    SPA is currently doing clean-up
  AND
      there exists a user-defined dynamic rule with a time limit
  AND
      the time limit has expired
THEN
  remove the dynamic rule
```

26. RULE-DELETE-UNUSED

```
IF
    SPA is currently doing clean-up
  AND
      there exists a FROB that is marked for deletion
    OR
      there exists a FROB that is no longer valid
THEN
  delete the FROB
```

27. RULE-DELETE-UNNAMED

```
IF
    SPA is currently doing clean-up
  AND
    there exists a PROCESS FROB whose NAME is undefined
THEN
  delete the PROCESS FROB
```

28. RULE-CHECK-HOSTHIST-STATUS

```
IF
    SPA is currently doing check-hosthist-status
  AND
    the name of the current host is the same as the host
    where hosthist is supposed to run
  AND
    hosthist is not running there
THEN
  start hosthist as a background task.
```

29. RULE-UPDATE-PROGRAM-HISTORIES

```
IF
     SPA is currently in the 'update-program-histories' phase
  AND
     the last time the program history program (proghist) was
     executed was over 24 hours ago
  AND
     the program history program (proghist) is available
THEN
  execute the program history program (proghist) to update the
  program history files
```

30. RULE-LOAD-PROGRAM-HISTORIES

```
IF
     SPA is currently in the 'load-program-histories' phase
  AND
     the program history program (proghist) is available
THEN
  load program histories from process history files
```

31. RULE-SAVE-SYSTEM

```
IF
   the shutdown process is running
THEN
   Save the system because a reboot will probably occur shortly.
   Saving the system will allow SPA to continue uninterrupted
   after the reboot.
```

32. RULE-CHECK-MAIL

```
IF
     SPA is currently checking for incoming mail messages
  AND
     there is mail to be read
THEN
   Read the incoming mail message(s). Create a MAIL_RECEIVED alert
   for each message that was read.
```

33. RULE-ABANDONED-PROCESS

```
IF
    SPA is currently analyzing data
  AND
    there exists a process ?proc
  AND
    the parent pid of ?proc is 1 (init)
  AND
    the user id of ?proc is not 'root' (uid = 0)
THEN
    ?proc is an abandoned process.
```

34. RULE-COLLECT-PROCESS

```
IF
    SPA is currently collecting process data
  AND
    there exists a host H
  AND
    H is the local host
  AND
    H is involved in a problem of some kind
  AND
    process data has not been collected this iteration
    of the data collection and analysis loop
THEN
    collect the process data on the host H
```

35. RULE-COLLECT-PROCESS-REMOTE

```
IF
    SPA is currently collecting process data
  AND
    there exists a host H
  AND
    the status of H is 'up'
  AND
    H is not the local host
  AND
    H is involved in a problem of some kind
  AND
    process data has not been collected this iteration
    of the data collection and analysis loop
  AND
    H is a remote host that SPA can investigate
THEN
    collect the process data on the host H
```

36. RULE-RENICED-PROCESS

```
IF
    SPA is currently analyzing data
  AND
    there exists a process ?proc
  AND
    the nice value of ?proc is > 0
THEN
    ?proc is a reniced process.
```

37. RULE-DIAG-PROCESS-MEM-A

```
IF
    SPA is currently looking for memory utilization problems
  AND
    there exists a process ?proc
  AND
    the RSS attribute of ?proc is larger than normal
    (Determining whether the RSS is larger than normal
     is accomplished by comparing it with the previous
     RSS attributes collected for this process.)
  AND
    ?proc is not involved in another problem of this kind
THEN
  create an instance of a MEM_SIZE problem
```

38. RULE-DIAG-PROCESS-MEM-B

```
IF
```

```
    SPA is currently checking for memory utilization problems
  AND
    there exists a process ?proc
  AND
    the RSS attribute of ?proc is larger than normal
    (Determining whether the RSS is larger than normal
     is accomplished by comparing it with the RSS
     obtained from accounting records.)
  AND
    ?proc is not involved in another problem of this kind
THEN
  create an instance of a MEM_SIZE problem
```

39. RULE-DIAG-PROCESS-TIME-USED

```
IF
    SPA is currently analyzing problems
  AND
    there exists a process ?proc
  AND
    the amount of time used by ?proc is longer than normal
    (Determining whether the time is longer than normal
     is accomplished by comparing it with the TIME_USED
     attribute obtained from accounting records.)
  AND
    ?proc is not involved in another problem of this kind
THEN
  create an instance of a TIME-USED problem
```

40. RULE-DIAG-PROCESS-ELAPSED-TIME

```
IF
    SPA is currently analyzing problems
  AND
    there exists a process ?proc
  AND
    the time that ?proc has been running is longer than normal
    (Determining whether the time is longer than normal
     is accomplished by comparing how long this process has
     existed with the ELAPSED_TIME attribute obtained from
     accounting records.)
  AND
    ?proc is not involved in another problem of this kind
THEN
  create an instance of a ELAPSED_TIME problem
```

41. RULE-DIAG-PROCESS-MEM-UTIL

```
IF
```

```
    SPA is currently checking for memory utilization problems
  AND
    there exists a process ?proc
  AND
    the percentage of memory used by ?proc is larger than normal
    (Determining whether the percentage is larger
     than normal is accomplished by comparing it
     with previous values of the MEM attribute for
     this process.)
  AND
    ?proc is not involved in another problem of this kind
THEN
  create an instance of a MEM_UTILIZATION problem
```

42. RULE-DIAG-PROCESS-CPU-UTIL

```
IF
    SPA is currently checking for cpu utilization problems
  AND
    there exists a process ?proc
  AND
    the percentage of the cpu used by ?proc is larger than normal
    (Determining whether the percentage is larger
     than normal is accomplished by comparing it
     with previous values of the CPU attribute for
     this process.)
  AND
    ?proc is not involved in another problem of this kind
THEN
  create an instance of a CPU_UTILIZATION problem
```

43. RULE-FIND-ZOMBIE-PROCESS

```
IF
    SPA is currently analyzing problems
  AND
    there exists a process ?proc
  AND
      the status of ?proc is that of a zombie process
    OR
      the flags of ?proc indicate that ?proc is in its
      final exit processing
  AND
    the amount of time used by ?proc is ?time_used
THEN
    Create an alert that a zombie has been found. Monitor
    ?proc to see if it is accumulating CPU time.
```

44. RULE-DELETE-PROCESS-INSTANCES

```
IF
    SPA is currently deleting instances of processes
  AND
    there exists a process P
  AND
    SPA has diagnosed a problem or alert involving P
  AND
    H is the only instance of a host or process
    involved in this problem.
THEN
  delete the instance of the problem or alert.
```

45. RULE-NFSD-PROCESSES

```
IF
    there exists a host H
  AND
    there are N 'nfsd' processes on H
  AND
    the load average on H is stuck at or within
    +/- 10% of N
  AND
    the 'nfsd' processes are all near the top the
    the list of processes in terms of CPU
    utilization
THEN
  create an instance of the NFSD_PROCESSES
  problem.
```

46. RULE-VALIDATE-PROBLEMS

```
IF
    SPA is currently seeking validation
    of its conclusions about problems
  AND
    there exists an alert or problem
  AND
      the status tells us it is not a problem
    OR
      the user tells us it is not a problem
THEN
  mark the problem for deletion
```

47. RULE-SHOW-PROBLEMS

```
IF
    SPA is currently looking for problem or alert instances
    of type ?type
  AND
    there exists a problem of type ?type
THEN
  display the problem
```

48. RULE-COLLECT-VM

```
IF
    SPA is currently collecting VM data on hosts
  AND
    there exists a host H
  AND
    SPA has diagnosed a problem on H
  AND
    H is up
  AND
    H is not the local host
  AND
    H is a host for which SPA collects VM statistics
THEN
  collect VM statistics on that host.
```

5 SPA Commands

ADVISE: Determine a strategy for resolving the specified problem(s).

The advise command causes SPA to determine appropriate actions to resolve the specified problem. One or more problems must be specified.

CDIR: Change current working directory.

The cdir command is used to change the current working directory to the specified directory. Syntax:

```
cdir [ dir ]
```

where `dir` is the new working directory. If no `dir` is specified, the working directory is changed to the home directory.

CHOST: Change or display the current host.

The chost command is used to change the current host to the specified host. The concept of the current host is similar to that of the current working directory. When referring to processes, if no host is specified, the current host is used. Syntax:

```
chost [ host ]
```

where `host` is the new current host. If no `host` is specified, the current host is displayed.

COLLECT: Collect data about hosts or processes.

The `collect` command is used to cause SPA to collect information about the system. Specific data types can be specified. `all` can be entered to collect all data types: `host`, `host-vm`, `host-tables`, and `process`. Examples:

```
collect host process
```

collects host and process data. Host information is collected from the `ruptime` utility and process data is collected from the `ps` utility.

```
collect
```

collects all data types: `host`, `host-vm`, `host-tables`, and `process`.

CONTINUE: Collect host and process data and then diagnose problems.

The **continue** command is used to cause SPA to collect information about the system and perform any necessary diagnoses. The setting of the **interactive** parameter (see the **set** command) affects how this command operates. If the interactive mode is enabled, the **continue** command causes SPA to run uninterrupted until it detects a situation requiring operator interaction. For example, when a performance problem is detected, SPA stops, displays the information about the problem, and waits for the user to take some kind of action on it. If the interactive mode is disabled, SPA runs indefinitely. The user must manually interrupt SPA (with Ctrl-C) and then enter:

```
:a
(spa)
```

in order to enter additional commands. If the interactive mode is set to a number, SPA loops that many times before stopping again and waiting for user input.

DATA-TYPES: Show valid SPA data collection information.

The **data-types** command can be used to display information regarding how SPA collects data from hosts and processes. The **data-types** command has two varieties:

```
data-types
```

Without any arguments, a list of the classes for which data is collected from the system is displayed.

```
data-types <class-list>
```

If one or more of the data collection classes are specified (in the **<class-list>** arguments), the utility name used to collect data from the system is displayed. Examples of data collection classes are **host**, **host-vm** and **process**.

DIAGNOSE: Display and/or update current performance problems.

The **diagnose** command is used display current problems. The status of problems can be updated with this command. Without any arguments, the **diagnose** command displays all problems. If an argument is specified, it must be the name of an existing problem or alert. The specified problems or alerts are displayed until you enter **10** or **quit**.

EXIT: Exit SPA to the Unix shell.

The **exit** command allows the user to exit SPA to the UNIX shell. Part of the exit processing includes updating the SPA statistics file. This file contains statistics about how many problems SPA has diagnosed and how many of them were valid.

HELP: Obtain a description of commands or SPA data types.

The **help** command can be used to obtain a description of commands. Zero or more command names can be entered as arguments. SPA data types (such as **host** or **process**) or system parameters (such as ***history-level***) can also be entered to get a description of them.

HISTORY: List the previous commands submitted to the system.

The **history** command can be used to review the commands submitted to the system. The current history file is in the directory from which the current SPA session was started. The filename is of the form:

```
cmd-history-MM-DD-YYYY
```

where

MM - the current month (1 <= *MM* <= 12)
DD - the day of the month (1 <= *DD* <= 31)
YYYY - the year

In addition to using the **history** command, this file can be viewed from the UNIX shell.

INCLUDE: Define user-defined rules for locating problems.

The **include** command is used to add a new problem type to the SPA system. The new problem type is described in terms of what knowledge base objects are involved (e.g. **host**), a relation that must be satisfied in order for an instance of the new problem type to be created (e.g. **load1 >= 9.5 and user_count < 4**), and a name for the new problem type.

Upon describing a new problem type with the **include** command, the user is asked whether there is a time limit for the new problem type. The time limit (if one exists) is the amount of time into the future that this problem type should be in effect. After that time, the problem description is removed from the system. If the user indicates that there is a time limit, he/she is asked to enter the length of time that the problem should be considered to be in effect. The time is entered in hours and minutes. For example, the time limit

```
12:30
```

means SPA will conduct a search for the problem for the next 12 hours and 30 minutes.

Syntax of the **include** command:

```
include from <class-list> where <search-cond> as <problem-name>
```

where

<class-list> - a comma-separated list of valid data classes. See the **types-of** command for a list of the valid data classes.
<search-cond> - a relation that must evaluate to true for all tuples of the resulting table.
<problem-name> - an identifier used as a name for the problem described in the **include** command.

PLOT: Data plotting.

The **plot** command is used to display a plot of the specified history data. Historical data for hosts and processes can be plotted. Only slots for which historical data is maintained can be plotted, however.

The plot command has the following syntax:

```
plot <type> <name> ( <slots> ) [ <count> ]
```

where <type> is a data class name (e.g. **host** or **process**),

<name> - is an instance of <type>,
<slots> - are valid slot names for <type>, and
<count> - is an optional data point count.

Once the data is plotted, the user enters a **gnuplot** session in which he/she may enter gnuplot commands. A **quit** (or **q**) terminates the session.

If the user wishes to obtain a hardcopy of the plot, the following commands can be entered in the **gnuplot** session:

```
set output "file"
set terminal postscript
replot
quit
```

where **file** is the name of the file to which to write the plot information. Note that the filename must be enclosed in double quotes. In this example, a postscript file is written. The resulting output file can be printed to a postscript printer with the **shell** command:

```
shell "lpr -P<printer> file"
```

where **<printer>** is the name of the printer to use.

PWDIR: Display the current working directory.

The **pwdir** command is used to display the current working directory. Syntax:

```
pwdir
```

There are no arguments to this command.

QUIT: Leave the SPA system.

Exit the SPA system. Control is returned to the Common Lisp environment. SPA can be re-entered by invoking the SPA system again with the following function call:

```
(spa)
```

RESET: Reset to initial conditions.

The **reset** command allows the user to re-initialize SPA to a state such that it appears that the user has just started the SPA session. This is a very dangerous thing to do. Invoke it only after careful consideration.

SAVE: Save the current state of the SPA system.

The **save** command is used to save the current state of the SPA system. A saved system can be restarted later for continuation of the current session. Once the SPA system is started there is no way to restore a different saved system state.

The syntax for the **save** command is as follows:

```
save [ <filename> ]
```

The **<filename>** is optional. The default is **save-spa** in the user's home directory.

SELECT: Relational knowledge base query.

The **select** command can be used to inspect the knowledge base using a syntax in the style of the SQL relational database query language. Queries have the following syntax:

```
select <slot-list> from <class-list> where <search-cond>
```

where

<slot-list> - one or more slot names defined for one of the classes in **<class-list>**. If more than one class in **<class-list>** has the same slot name, the slot name can be uniquely identified by using a *range variable.*

For example, the following query involves two classes with a **name** slot:

```
select P.name,' running on ',H.name from host H,process P
    where P.host = H.name
        and ( H.name = 'cs' or H.name = 'jensen' )
```

This query results in a table containing the names of all processes running on **cs** or **jensen**. The names of processes and hosts are separated by the string " running on ".

<class-list>	-	a comma-separated list of valid data classes. See the **types-of** command for a list of the valid data classes.
<search-cond>	-	a relation that must evaluate to true for all tuples of the resulting table.

<search-cond> is one of the following forms:

```
<slot-val> <op> <slot-val>
<slot-val> <op> <search-cond>
not <search-cond>
<search-cond> <bool> <search-cond>
```

where

<slot-val>	-	a *slot* or a *val*
<op>	-	one of: =, <>, <=, <, >=, >
<bool>	-	one of **and** or **or**
slot	-	a slot name defined for one of the classes in **<class-list>**
val	-	a value type-compatible with *slot* (if *val* is a string, enclose it in single quotes)

The **not** keyword provides the negation of the specified relation. Thus,

```
select a,b,c from c1 where not (a = 14 or b > 0)
```

is equivalent to

```
select a,b,c from c1 where a <> 14 and b <= 0
```

Complex queries can be formed with the **union** of two or more **select** commands. Usually, however, a union of two or more queries has an equivalent single query form that is more understandable and involves less typing.

SET: Modify SPA system parameters.

The **set** command is used to modify system parameters. The syntax is:

```
set <parameter> = <value>
```

Currently, the only parameters that are supported are

```
history debug prompt interactive
```

Accepted values for **history** are:

0 - history is disabled
1 - command history is maintained
2 - command and action history is maintained
3 - history level 2 plus diagnostic output is provided

Setting **history** to **on** enables the command history (level 1).

Setting **history** to **off** disables the command history (level 0).

Setting **debug** to **on** enables diagnostic output.

Setting **debug** to **off** disables diagnostic output.

Setting **interactive** to **on** (or **T**) means SPA stops and queries the user upon locating a problem.

Setting **interactive** to **off** (or **nil**) means SPA does not stop and query the user upon locating a problem. Rather, it continues the process of data collection and data analysis. the user must interrupt SPA manually with **Ctrl-C**.

The **prompt** can be set to any string. By default, the prompt consists of a string in the following form:

```
SPA(<modes>)>
```

where **<modes>** is an indication of the current settings of the **history** and **debug** modes described above. Within the **<modes>** list,

```
H=D
```

indicates that the history level is set to the value **D**. If the debug mode is active, thisis indicated by the string **debug** within the **<modes>** list. If the prompt string is redefined, the modes are no longer be displayed. The current modes can still be displayed by entering

```
set
```

This displays all current system parameter settings. Or, the prompt string can be returned to its default behavior by entering:

```
set prompt = default
```

Other parameters can be set to any value. If a parameter is set to a string value representing a SPA command, it can then be used in a way that is similar to csh environment variables. For example,

```
set foo = "cs jensen shafer bliss acme"
```

then entering

```
show $foo
```

shows the information for **cs**, **jensen**, **shafer**, **bliss**, and **acme**. Something like this:

```
set bar = "select name from host where load1 > 2.0"
```

causes something like this:

```
$bar
```

to execute the above **select** command.

SHELL: Execute a program in a UNIX shell.

The **shell** command is used to execute a program. The only argument to the **shell** command is a string describing the program to execute.

NOTE: Currently, the **shell** command is not fully interactive. The **shell** command can only be used to provide a display of the output of the program that was executed. **DO NOT** run programs that require input from the user (such as an editor). Input to some programs could be provided by redirection, however.

SHOW: Display information about hosts, processes, problems, or rules.

The **show** command is used to display information regarding instances of knowledge base instances. Usually, the instances are hosts, processes, or problems, but any defined knowledge base instance could be specified. The name(s) of one or more instances must be specified. The syntax of the **show** command is as follows:

show [<type>] <name> [<name-list>]

<type>	-	one of: **all**, **dynamic**, **history**, or **rule**. These can be abbreviated to one or more characters.
<name>	-	can be: (1) the name of a SPA object—a host, process, or problem instance, (2) the name of a SPA object class—usually specified as **{class host}**, **{class process}**, etc., (3) a rule name. Which of these apply depends on the **<type>** argument.
<name-list>	-	a whitespace-separated list of **<name>** as described above.

Examples:

show <instance> [<instance-list>]

This is the simplest form of the **show** command. It is used to obtain information about one or more knowledge base instances. For example,

show asylum cs peruvian

displays knowledge base information on the hosts called **asylum**, **cs**, and **peruvian**.

show all <class-name> [<class-list>]

This command shows the names of all instances of **<class-name>** that SPA knows about. For example,

show all host

shows all names of hosts. Other accepted class names are:

host process problem alert

program_history process_history host_history

A list of all defined data-types can be obtained with the **data-types** command.

show hist <instance> [<instance-list>]

This command displays the historical information on the **<instance>** specified. Each occurrence of **<instance>** specified must be a defined knowledge base object.

show rule <rule-name> [<rule-name-list>]

This command displays the source code for the specified SPA diagnostic rule.

show dynamic-rule <rule-name> [<rule-name-list>]

This command displays the source code for the specified SPA user-defined rule (Defined with the **include** command.).

SLOTS-OF: Show valid slot names.

The **slots-of** command returns the names of slots for the specified data types. As many data as desired can be specified. Data types are **host**, **process**, **host_history**, **process_history**, **program_history**, **statistics**, etc.

STATUS: Display SPA performance statistics.

The `status` command is used to display statistics regarding the performance of SPA. The display includes information about how many problems SPA discovered and how many of those problems were determined by the user to be valid.

TYPES-OF: Show valid SPA data class names.

The `types-of` command returns the currently defined class names that inherit from the specified class. As many class names as desired can be specified.

UPDATE: Update host and process history information.

The `update` command is used to update the historical profile of a host or process. This could be useful if the data SPA is currently using is corrupted in some way.

WHY: Explain a problem.

The `why` command is used to query the SPA system about how it diagnosed the specified problems as being problems. A description of the problem is given along with a description of the context in which SPA found the problem.

?: Display a summary of all commands available.

Entering a ? or `commands` provides a summary of all commands available.

6 Maintaining SPA

The following is a general guide to assist a system administrator in modifying the behavior of SPA to suit the needs of a particular installation. It is not a step-by-step how-to guide with all the details, but rather a conceptual level guide to get a system administrator started in the right direction in making modifications to SPA.

6.1 Customizing SPA With the Initialization File: `.sparc`

If there exists a file called `.sparc` in the user's home directory, SPA reads this file for initialization information. Some of the information in this file is required in order for SPA to operate correctly. Other information is optional and can be used to customize the behavior of SPA.

6.1.1 Required Information

The UNIX utilities that SPA invokes to collect data must be defined. This is done through the `define-data-type` function calls. The usage of the `define-data-type` function is defined in Section 6.2.

6.1.2 Optional Information For Customizing SPA

Setting system parameters is the primary way of modifying the behavior of SPA. System parameters are set by using the `set-parameter` macro. Built-in system parameters are defined in Section 7.

6.2 How to Add a Monitored Statistic to SPA

The following is a general guide to adding a monitored statistic in the SPA expert system. It is not a step-by-step guide because adding a statistic is a complex procedure and the specific steps involved depend on the exact statistic desired and what utilities are involved in collecting the data.

The following description outlines the files involved in adding a statistic to the **host** class. The description is similar for a statistic for the **process** class. Each paragraph below outlines changes necessary to a particular source file of SPA.

Changes Required to .sparc

Describe what utility(ies) are used to gather the new statistic. This is done with a call to the **define-data-type** function. The syntax of the **define-data-type** function is as follows:

(**define-data-type** *dtype dtypeClass utility arguments fields*)

where

dtype is the name of the data type. When this process is complete, you should be able to get SPA to collect the new statistic via the **collect** command. For example, if you want to define a new statistic called **host-mem** that will collect information regarding the real and available memory on a host, then **host-mem** is the name of the data type.

dtypeClass is the FROBS class that the data is stored in. In this case, the class **host**.

utility is the utility used to collect the information[4].

fields is a description of what slots of the instances of *dtypeClass* are updated from what columns of the output of *utility*. The format of *fields* is as follows:

$$(desc_1\ desc_2\ \ldots\ desc_N)$$

where each $desc_i$ for $1 \leq i \leq N$ is of the form:

$$(slot\ (type\ start\ end))$$

where

slot is a slot name of class *dtypeClass*.

type is the type of data to expect. Valid types are:

- number
- string
- symbol

[4]IMPORTANT: The name of the host must appear somewhere in the data. This is so the data collection routines of SPA can find the proper FROBS instance to store the information. Also, the information collected must be contained in a single record. If the output of the utility is multiple records (like **ruptime**), then there must be one record for each host. If there are multiple records for the same host, then the same data (described in *fields*) is collected from each record.

start is the starting column number to search.

end is the ending column to search. *end* can be a number, a character, or the special symbol `eos`. `eos` means the end of the current input record. A character means search until a matching character is found. For example, `(name (string 0 #\space))` gets the `name` slot as a string starting from the beginning of the input record until a space is encountered.

Changes Required to `hosts.l`

1. Add the slots that appear in *fields* to the class definition for hosts.
2. If statistics on the new slots are to be maintained, add the same slot to the host-history class.
3. Add the proper kind of statistic to the host-history instance creation method. For example, suppose a slot called `sr` is added to the host class (and, by step 2, also to the `host-history` class). Then, add to the `make-spa-object` method for the host-history class an S-expression of the form

   ```
   (setf (sr hist) (make-spa-object {class statistics} name))
   ```

   where `hist` is a local variable representing the new host-history instance and `name` is the name of the host. If the statistic is supposed to be a time-series model, then an S-expression of the form

   ```
   (setf (sr hist) (make-spa-object {class time_series_model} name))
   ```

   is added.
4. Add S-expressions to save the current value of host data in the corresponding host-history instance. This is done in the daemon called `gather-host-stats`.
5. If the new slot contents should be displayed when a `show` command is entered, add the new slot to the `print-spa-object` method for *dtypeClass*.

Changes Required to `host-rules.l`

1. Add rules of inference for the new statistics. This is optional.
2. If the new rules are to diagnose a new type of problem, you will want to make the type of problem a FROBS class. The remaining steps need to be done only if a new problem class is created.

Changes Required to `classes.l`

Add the new problem class. Use other class definitions as a guideline. Don't forget to document the new class as the other classes have been documented. This file is scanned to create documentation that is used at run-time.

Changes Required to `globals.l`

Add the name of the new problem class to the global variable `*spa-problem-types*`.

7 Customizing SPA

Setting system parameters is the primary way of modifying the behavior of SPA. System parameters are set by using the **set-parameter** macro. The **set-parameter** macro is called in the following way:

(set-parameter *'ParamName Value*)

where *ParamName* is a name of a defined system parameter and *Value* is a new value. System parameters can also be set from the SPA main loop using the **set** command. For example,

set *ParamName = Value*

sets the system parameter indicated by *ParamName* to the value indicated by *Value*.

The next paragraphs describe the system parameters that are defined and how they can be used.

DYNAMIC-HOSTS The ***dynamic-hosts*** variable defines whether SPA dynamically creates host instances as it encounters them. To set ***dynamic-hosts*** to **NIL** means no, it won't (useful when you only want to monitor a fixed number of hosts). To set ***dynamic-hosts*** to **T** means SPA creates instances of all hosts it encounters in a **ruptime** output. The user should set this variable if he/she wants a behavior other than the default. The default is to have ***dynamic-hosts*** set to **NIL**.

DYNAMIC-PROGRAMS The ***dynamic-programs*** variable defines whether SPA dynamically creates program histories as it encounters them. To set ***dynamic-programs*** to **NIL** means no, it won't (useful when you only want to monitor a fixed number of programs). To set ***dynamic-programs*** to **T** means SPA creates instances of program histories for all programs it encounters. The user should set this variable if he/she wants a behavior other than the default. The default is to have ***DYNAMIC-PROGRAMS*** set to **NIL**.

HIGH-LA-LIMIT This variable defines an upper limit used by the load average diagnostic rules. If the load averages are above ***high-la-limit***, a load average problem instance is always created.

HISTORY-LEVEL The ***history-level*** variable indicates how much of the output of the SPA program to keep in its history files. Setting ***HISTORY-LEVEL*** is equivalent to setting the **history** system parameter using the SPA **set** command. For example, the following **set-parameter** expression

```
(set-parameter '*history-level* 3)
```

is equivalent to the following SPA set command:

```
set history = 3
```

Accepted values for **history** are:

- 0 - history is disabled
- 1 - command history is maintained
- 2 - command and action history is maintained
- 3 - history level 2 plus diagnostic output is provided

Setting **history** to **on** enables the command history (level 1).
Setting **history** to **off** disables the command history (level 0).

HISTORY-IN-MEMORY The ***history-in-memory*** variable indicates whether a history of readings is kept in memory or not. If ***history-in-memory*** is not asserted (is set to **NIL**), no readings are kept in memory. The history of readings must be retrieved from disk using the **hosthist** utility.

The purpose of this variable is to assist SPA in saving memory. Set this variable (to **T**) only if you have sufficient heap space to allow SPA to keep ***sdev-reading-count*** (see **help *sdev-reading-count***) readings for all the statistics that SPA maintains.

HOST-MODEL-TYPE The ***host-model-type*** variable indicates what type of modeling to use. This variable must be set to either

```
'STATISTICS
```

or

```
'TIME_SERIES_MODEL
```

TIME_SERIES_MODEL is the default.

INTERACTIVE When ***interactive*** is asserted (set to **T**), SPA stops and queries the user on all problems/alerts that are found. If ***interactive*** is not asserted (is set to **NIL**), SPA does its best to continue to locate new problems and remove outdated ones.

LOW-LA-LIMIT This variable defines a lower limit used by the load average diagnostic rules. If the load averages are below ***low-la-limit***, no diagnostic rules are invoked.

MAILABLE-USERS This variable defines what users SPA sends mail to in the event that SPA decides a message regarding an executing process owned by the user should be sent.

N-LIMIT This variable defines the number of readings of a statistic that are collected before any diagnoses are attempted involving the statistic.

PROMPT-STRING The variable ***prompt-string*** defines the prompt that SPA uses. Setting this variable is equivalent to using the SPA **set** command to set the **prompt** system parameter. For example, the following **set-parameter** expression

```
(set-parameter '*prompt-string* "INPUT > ")
```

is equivalent to the following SPA **set** command:

```
set prompt = "INPUT > "
```

The **prompt** can be set to any string. By default, the prompt consists of a string in the following form:

```
SPA(<modes>) >
```

where <modes> is an indication of the current settings of the **history** and **debug** modes (described in the **set** command documentation). Within the <modes> list,

```
H=D
```

indicates that the history level is set to the value **D**. If the debug mode is active, this is indicated by the string **debug** within the <modes> list. If the prompt string is redefined, the modes are no longer displayed. The current modes can still be displayed by entering

```
set
```

This displays all current system parameter settings. Or, the prompt string can be returned to its default behavior by entering:

```
set prompt = default
```

RSH-HOSTS The ***rsh-hosts*** variable defines what remote hosts can be accessed (via **rsh**) to gather additional information.

SDEV-READING-COUNT This variable defines the number of readings of a statistic used in calculating the standard deviation for that statistic.

SPA-DEBUG The ***spa-debug*** flag is used for providing diagnostic output along with the regular program output. The output is voluminous so setting this value to **T** should be undertaken with care. Setting this variable to **T** is equivalent to executing the following SPA command:

```
set debug = on
```

STD-DEV-LIMIT Readings that are more than ***std-dev-limit*** standard deviations above the average reading or more than ***std-dev-limit*** standard deviations below the average reading can be considered abnormal readings. Abnormal readings can cause the creation of problem instances.

VM-SAMPLES When virtual memory statistics are gathered on a host, the **vmstat** utility is invoked. This number represents the iterations of the **vmstat** utility that are performed. Iterations generally are performed every 5 seconds.

7.1 Defining Host Profiles

There is some information about hosts that SPA cannot determine at runtime by itself. The **host-profile** function allows the user to define what hosts are to be monitored and what the primary usage of the host is.

The **host-profile** function has the following syntax:

(**host-profile** *hostName userProfile hostClass*)

The *hostName* argument is a string indicating the name of the host. The *userProfile* argument is a list indicating what kind of users are typically on the system. Possible choices for this argument are shown in Table 4. More than one *userProfile* can be chosen for a host. The *hostClass* argument is optional and indicates what the intended primary usage of the host. Possible choices for this argument are described in Table 5 and shown in Figure 4.

7.2 Defining Program Profiles

The **program-profile** function allows the user to define what programs are to be monitored and what the general classification of the program is.

The **program-profile** function has the following syntax:

(**program-profile** *programName programClass*)

The *programName* argument is a string indicating the name of the program to monitor. The *programClass* argument is optional and indicates the general classification of the program. Possible choices for this argument are described in Table 6 and shown in Figure 5.

7.3 Other parameters

The value of any defined Common Lisp symbol can be set with the **set-parameter** macro. For example, the **lisp::*verbose-gc*** symbol, which indicates whether garbage collection messages should be printed or not, can be set. The following S-expression prevents the messages from being printed:

```
(set-parameter 'lisp::*verbose-gc* nil)
```

Table 4: Defined User Profiles

Name	Description
`student`	Users are typically undergraduate students
`staff`	Users are typically staff members doing software development work
`grad-student`	Users are typically graduate students
`research`	Users are typically staff members, students, or professors doing research work

Table 5: Defined Host Classes

Name	Description
`fileserver`	host is used as a fileserver
`general`	general multi-user usage
`research`	research
`workstation`	single user usage

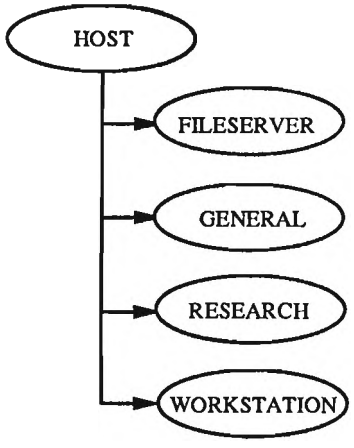

Figure 4: Host Type Hierarchy

Table 6: Program Types Defined in SPA

Name	Description
`application`	a program written by a user
`editor`	an editing or word-processing utility
`language`	a language processor
`compiler`	a language processor that is a compiler
`interpreter`	an interpreted language processor
`system-process`	a system process such as `sendmail`
`unix-utility`	a UNIX shell utility such as `grep`
`unix-daemon`	a UNIX daemon process such as `rlogind`

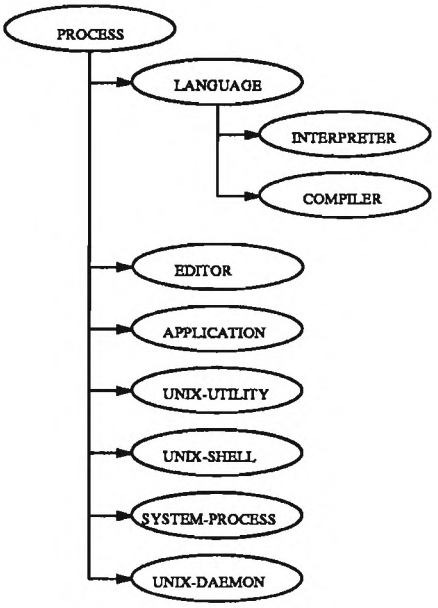

Figure 5: Program Types Defined in SPA