Satool - A System Administrator's Cockpit

Todd Miller – University of Colorado, Boulder
Christopher Stirlen – University of Colorado, Boulder
Evi Nemeth – University of Colorado, Boulder

ABSTRACT

Monitoring large numbers of machines in a distributed environment can be time consuming and inefficient. Often a system administrator finds there is a problem with one of the machines or network links by a phone call from a frustrated user. **Satool** provides a way to efficiently monitor groups of machines and find problems and potential problems quickly; it's sort of an early warning system for sysadmins.

Satool is composed of three independent parts: an SNMP (Simple Network Management Protocol) agent that runs on each machine to be monitored, a database that collects data from each client machine, and a graphical user interface (GUI) that acts as the interface between the user and the database.

Introduction

With the advent of fast, inexpensive workstations, the standard computing environment has changed from a small number of vaxen to a distributed network that has become increasingly complex and difficult to administer. Monitoring large numbers of machines in a distributed environment can be time consuming and inefficient. In such an environment, it is usually not possible to see the warning signs that point to imminent disaster for a machine or network. Instead, system administators find themselves fighting fires when their time would be better spent elsewhere. Satool provides a way to efficiently monitor groups of machines and find problems and potential problems quickly in a straightforward manner.

In designing **satool** we have tried to adhere to two fundamental principles: remain totally configurable (outguessing a sysadmin's needs is hopeless) and reuse existing tools where possible. On the data gathering side, configurable items include: the actual data to be collected, how often to collect it, threshold values that indicate a problem, the action to take if a threshold is exceeded. Configuration for the GUI selects the hosts or devices to display, the type of widget to represent a variable, and personal preferences for items like the arrival of an alarm condition (for example color, sound, blinking, etc). We use the **tcl** and **tk** languages from John Ousterhout at the University of California, Berkeley for the display. An SNMP agent and sysadmin MIB (Management Information Base) provide for communication between the data collection and data gathering activities. The actual data gathering is done by an SNMP agent running on the hosts being monitored. We have extended the CMU SNMP 1.1b agent and MIB to include variables of interest to UNIX system administrators not included in the standard network or host MIBs. SNMP agents update a variable when its value is requested. Thus, instead of the agent sending information to the server at regular time intervals, the server polls the agent when it wants a new value of a variable. (The one exception is when the agent first starts up. It sends a trap to the server to alert it that the agent is alive and can now be monitored). Since the responsibility to request data falls on the server, the server has complete control over what data it requests from the agent and how often it requests it.

The **satool** server maintains a database of client machines and the values of supported variables gathered during the last poll. Normally, a client is added to the database when the server receives a message from the SNMP trap daemon that the client's SNMP agent is running. However, the server also maintains an on-disk copy of the database that is read in on invocation of the server. Because of this, the database will survive system crashes. Client machines can also be specified in the server's configuration file to "prime the cache" of machines to monitor. The time between polls is configurable on a per-machine basis (overriding a stated default). The server listens for database requests from the GUI on port 0xFB1 (that's a one not an I).

The **satool** display system includes an X based GUI (Graphic Users Interface) built using **tcl/tk.** It supports a hierarchical top level view of the machines being monitored. Machines can be grouped to allow the screen area to scale with the number of machines being watched. For example, a sysadmin can configure his workstation to display the machines he is directly responsible for, the machines that his colleague on vacation is responsible for, the printer down the hall that he uses regularly, and his network gateway to the outside world. After traversing the hierarchy to an individual machine, the display contains

visual widgets representing the health of that machine: disk space free, memory statistics, cpu activity, mail queue length, nfs statistics, etc. The user can choose the form of the widget (currently a number, a thermometer display, or a sliding window histogram) to display each quantity.

Alarm conditions for each variable can be expressed in the configuration files as well. If the value of a variable exceeds the alarm threshold, a special action (blinking, beeping, reverse video, digital pager, etc.) is taken on the screen to indicate it. Alarms are propagated to the top level display, thus if /var/tmp fills up on host *heineken* on the *beers* subnet of the *cs* domain and if the groups are set appropriately, the alarm condition would be noted in the widgets representing *heineken*, *beers*, and the *cs* domain. A user would see or hear the alarm independent of which level he was actively displaying at the time.

A working prototype of **satool** exists. It will be used this fall by the Computer Science Department's undergrad lab sysadmin group and graduate/faculty research sysadmin group to monitor about 300 machines. We expect to use feedback from these groups to expand the sysadmin MIB and to build more display widgets. The code will be freely available with a Berkeley style copyright notice.

Satool Daemon (satoold)

Satoold has three main functions: gather data from its clients, store data in a database, and service requests to access that data from the GUI.

Data Gathering

Satoold polls clients for data via SNMP at configurable intervals. All polling is done by a forked process which passes data back to the parent via a UNIX domain socket. Polling times for clients are kept in a linked list (here referred to as the "timer queue" although it is not truly a queue) sorted by time to poll (in UNIX time format). To allow for concurrent polls, a variable may be set at compile-time, specifying the number of polling processes allowed. A counter is used to keep track of the number of polling children along with an array of their process ids.
The flow of control is as follows:

- **Satoold** is notified by the SNMP trap daemon that a client has come up.
- The client is inserted into the timer queue if it is not already present there.
- If the client is not already in the database, it is added. Otherwise, the client's current database entry is updated to reflect the fact that the client is now up.
- An alarm goes off, signifying that it is time to poll a client.
- A signal handler is called, and a child is forked to poll the client.
- The child effects the poll and sends the data back to its parent via a UNIX domain socket.
- If the connection to the client timed out, that machine is now marked as down and its polling frequency is reduced.

Data Storage

Data about clients is stored in an *ndbm*(3) database, keyed on the fully qualified hostname. The use of *ndbm*(3) allows for some basic crash recovery.

- The old database is read on invocation of **satoold** and a timer queue is created based on the information in the database.
- Obviously bogus (empty) keys are discarded (this is the most common cause of database corruption we have seen).

Servicing GUI Requests

Satoold accepts connections on port 4017 (0xFB1 in hex). Connections time out after five minutes of inactivity. The protocol used is similar to *sendmail's.* The protocol commands are:

HELO	Say hello to the daemon.
HELP	Prints a short help message.
LIST	Lists the all clients in the database.
GET	Gets the data for a particular machine.

Satoold responds to each command with a three digit completion code and a status/error message (in text) before returning the requested data (also like *sendmail).*The

First Digit:

2	Command completed
5	Command failed with a fatal error

Second Digit:

0	Syntax
1	Information
2	Connection
3	Host
5	Data

Third Digit:

The third digit is used to differentiate between codes that have the same first two digits. Ie: code 220 is the "connection established" greeting, and 221 is the "connection closed" message.

A session looks like this:
*** insert session here. 40 columns is just too painful, so use a figure ***

SNMP Agent

The SNMP (Simple Network Management Protocol) agent used in **satool** is based on CMU snmp1.1b from Carnegie-Mellon University. This release is MIB-I compliant. There are two major differences bewteen stock CMU snmp1.1b and the **satool** version: a configuration file and support for **satool** variables in the MIB (Management Information

Base).

Config File

The **satool** SNMP agent's configuration file is currently only used to specify a host to send coldstart traps to. On invocation, the agent will send a colstart trap to the host listed in the config file (if that file exists).

Satool Variables

The agent now supports variables defined by the **satool** MIB (see next section for the MIB). The values for most of the new variables are obtained via "helper scripts" that run standard UNIX commands and parse the output into a form that the agent can use. Their are two major reason to use these "helper scripts" (in accordance with our design goals). The first is the increased portability and flexibility scripts provide. The second is the desire to use existing UNIX tools where available.

There is, however, a large problem with the approach above; it is extremely slow for a large number of variables. The reason for this is that for each variable the agent does a *popen*(3) call which forks and execs the script. The solution is to have the script output all the variables we might be interested in at once and cache the values. Ie: instead of calling a *vmstat*(1) helper script eighteen times, we call it once and cache the values for ten seconds. Subsequent requests for any of the variables will get the cached values. This speeds things up considerably and the ten second granularity is considered acceptable.

Satool MIB

```
-- sa tool

cu OBJECT IDENTIFIER
    ::= { enterprises 632 }

satool OBJECT IDENTIFIER ::= { cu 1 }

-- to get real load ave. divide the
-- int by 100
satoolLoadAve OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 1 }

-- number of entries in the mail queue
satoolMailQueueLen OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 2 }

-- number of system calls / sec.
satoolNumSysCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { satool 3 }

-- process information
processes OBJECT IDENTIFIER
    ::= { satool 4 }

-- number of processes
satoolNumProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 1 }

-- number of processes in disk wait
satoolNumWaitingProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 2 }

-- number of zombied processes
satoolNumZombieProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 3 }

-- number of processes in the run queue
satoolRunQueueLen OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 4 }

-- number of processes blocked for
-- resources
satoolNumBlockedProcs OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 5 }

-- number of processes runnable
-- but swapped
satoolNumRunnableButSwapped OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { processes 6 }

-- vm information
vm OBJECT IDENTIFIER ::= { satool 5 }

-- number of context switches / sec
satoolNumContextSwitches OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
```

```
    STATUS optional
    ::= { vm 1 }

-- number of active virtual pages
satoolActivePages OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 2 }

-- number of free virtual pages
satoolFreePages OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 3 }

-- number of page reclaims / sec
satoolPageReclaims OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 4 }

-- number of pages attached / sec
satoolPagesAttached OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 5 }

-- number of pages paged in / sec
satoolPageIns OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 6 }

-- number of pages paged out / sec
satoolPageOuts OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 7 }

-- number of pages freed / sec
satoolPagesFreed OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 8 }

-- anticipated short term memory
-- shortfall
satoolMemLow OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 9 }

-- number of pages scanned clock
-- algorithm / sec
satoolPagesScanned OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { vm 10 }

-- io stuff
io OBJECT IDENTIFIER ::= { satool 6 }

-- number of device interupts / sec.
satoolNumInterupts OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { io 1 }

-- rpc stuff
rpc OBJECT IDENTIFIER ::= { satool 7 }

-- number of rpc calls (server)
satoolServerRpcCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 1 }

-- number of bad rpc calls (server)
satoolServerRpcBadCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 2 }

-- number of empty rpc calls (server)
satoolServerRpcNullRecv OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 3 }

-- number of rpc calls with too small
-- a body (server)
satoolServerRpcBadLen OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 4 }

-- number of rpc calls that failed to
-- decode into xdr (server)
satoolServerRpcXdrCall OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 5 }

-- number of rpc calls (client)
```

```
satoolClientRpcCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 6 }

-- number of bad rpc calls (client)
satoolClientRpcBadCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 7 }

-- number of retransmitted rpc calls
-- (client)
satoolClientRpcRetrans OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 8 }

-- number of rpc calls where the reply
-- transaction ID did not match the
-- request transaction ID (client)
satoolClientRpcBadXid OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 9 }

-- number of rpc calls that timed out
-- (client)
satoolClientRpcTimeOut OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 10 }

-- number of times the client had
-- to sleep
satoolClientRpcWait OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { rpc 11 }

-- nfs stuff
nfs OBJECT IDENTIFIER ::= { satool 8 }

-- number of nfs calls (server)
satoolServerNfsCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 1 }

-- number of bad nfs calls (server)
satoolServerNfsBadCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 2 }

-- number of nfs calls (client)
satoolClientNfsCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 3 }

-- number of bad nfs calls (client)
satoolClientNfsBadCalls OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 4 }

-- number times a client structure
-- was successfuly gotten
satoolClientNfsNclGet OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 5 }

-- number times all client structures
-- were busy
satoolClientNfsNclSleep OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { nfs 6 }

-- cpu stuff
cpu OBJECT IDENTIFIER ::= { satool 9 }

-- percent of cpu in user time
satoolCpuUserTime OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { cpu 1 }

-- percent of cpu in system time
satoolCpuSysTime OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { cpu 2 }

-- percent of cpu in idle time
satoolCpuIdleTime OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { cpu 3 }

-- percent of cpu spent running niced
```

```
-- processes
satoolCpuNiceTime OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS optional
    ::= { cpu 4 }

-- table from df
satoolDfTable OBJECT-TYPE
      SYNTAX  SEQUENCE OF DfEntry
      ACCESS  read-write
      STATUS  optional
      ::= { satool 10 }

dfEntry OBJECT-TYPE
      SYNTAX  DfEntry
      ACCESS  read-write
      STATUS  optional
      ::= { satoolDfTable 1 }

DfEntry ::= SEQUENCE {
  dfIndex
      INTEGER,
  dfDevice
      OCTET STRING,
  dfMountPoint
      OCTET STRING,
  dfTotalKb
      INTEGER,
  dfUsedKb
      INTEGER,
  dfAvailKb
      INTEGER,
  dfCapacity
      INTEGER
}

dfIndex OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 1 }

dfDevice OBJECT-TYPE
      SYNTAX  OCTET STRING
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 2 }

dfMountPoint OBJECT-TYPE
      SYNTAX  OCTET STRING
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 3 }

dfTotalKb OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 4 }

dfUsedKb OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 5 }

dfAvailKb OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 6 }

dfCapacity OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      ::= { dfEntry 7 }
```

Supported Architecturres

Currently, the SNMP agent is only known to work under 4.3 BSD. The **satool** elements of the agent are known to also work under Ultrix 4.2/4.3 and should be easily portable to most versions of UNIX. The **satool** GUI will run on any system capable of running **tcl/tk. Satoold** should run on any version of UNIX that supports Berkeley sockets and *ndbm* (3) .

Future Enhancements

There is still work to be done on **satool.** There are plans to add the following functionality MIB-II support Support for more architectures in the SNMP agent Extra modules to alert sysadmins of pending/existing problems (email and pager).

Author Information

Todd Miller is a recent graduate of the University of Colorado, Boulder where he receieved a BS-CS and served as a systems administrator for two of the four years he spent there. He is currently unemployed and can be found playing guitar for change on the Pearl Street Mall in Boulder. Reach him electronically at millert@cs.colorado.edu or uunet!boulder!millert.

Christopher Stirlen is a masters student in Computer Science at the University of Colorado, Boulder. Graduation date is yet unknown and he plans to delay reality for as long as possible. Reach him electronically at stirlen@cs.colorado.edu or uunet!boulder!stirlen.

Evi Nemeth is an Associate Professor ay the University of Colorado, Boulder. Reach her electronically at evi@cs.colorado.edu or uunet!boulder!evi.