Gathering statistics from servers

Iain Rae
Division of Informatics

iainr@dcs.ed.ac.uk

Some notes on using lm_sensors, snmp and nrg/rrdtool on gathering stats from Linux (and other) servers.

Table of Contents

Overview

Motherboards these days usually come with some sensors which can log basic temp/voltage and other data, the lm_sensors[1] package allows linux to drop this information and some basic information about other hardware (e.g. how many and what kind of DIMMS are installed) into /proc. This data is then accessible either though commands on the pc or via snmp using a script to extend the snmp agent[2]. Using something like MRTG[3],NRG[4] or rrdtool[5] it's fairly easy to chart this data over time (or we could just log it).

If we are using snmp then it would also be possible to use the trap machanism to raise alarms when systems ran over temperature or out of disk space etc.

The lm_sensors package

This is a set of kernel modules and a couple of scripts which are based around the ISA and i2c/smbus bus and originally the lm78 sensor chip though a number of other bits of hardware are now supported. The full range of supported hardware can be found at the website[6].

A quick tour of the lm sensors modules

kernel/module versions

The first thing to bear in mind is that the modules will only work with the specific kernel they have been compiled against, this makes building and maintaining the rpms a bit messy. Also in order to compile lm_sensors you need to install a matching set of i2c kernel source patches (well replacement header files). Given that this is all hopefuly going to go away in the 2.4 kernel and so that we don't have to rebuild the kernel every time we want to make a change to lm_sensors we will run with the current kludge. Once the lm_sensors rpms have been built they can be installed in the usual manner.

Currently the lm_sensors 2.5.5-4 rpms are built against 2.2.17_server-0.4. With 2.6.0 the sensors rpm's will reflect the kernel version in the rpm name so we will have lm_sensors-2.6.0_2.2.17_server_0.4-1.rpm which is really ugly but I can't see a better way of doing it and 2.4 is going to come along and save the day anyway.

Support is currently only for the server kernel, mostly to keep things a bit simpler.

What modules do what, where do I get them (and what do I have to load)

The sensors package is built on a roughly 3 high stack of modules. The top layer provides hooks into the kernel, the second layer provides support for the busses and /proc, finally the lowest layer provides support for the specific hardware you are running, if we take a look at an lsmod on a motherboard which uses the via686 chipset

Example 1. Output from lsmod.

```
[callisto]root: lsmod
Module                  Size  Used by
usbcore                45552   0  (unused)
eeprom                  3008   0  (unused)
```

```
via686a                 8656   0
sensors                 5728   0  [eeprom via686a]
i2c-isa                 1168   0  (unused)
i2c-viapro              3568   0  (unused)
i2c-core               11424   0  [eeprom via686a sensors i2c-isa i2c-
viapro]
autofs                  9168   1  (autoclean)
3c59x                  22128   2  (autoclean)
```

Working from the top of the stack,

- *i2c-core* Base hooks into the kernel..
- *i2c-viapro* module to support isa/i2c buses found on via VT82C596,VT82C596B, and in this case VT82C686A/B chipsets. NB this doesn't guarentee you can talk to the sensors, just the bus.
- *i2c-isa* supports the Isa bus, yes even if you don't have any ISA slots in the PC that good old ISA bus is still going strong.
- *sensors* provides access to the proc filesystem, specifically /proc/sys/dev/sensors.
- *via686a* specific support for the sensor hardware shipped with the via686a (will work with b).
- *eeprom* provides information about memory installed.

Running /usr/sbin/sensors-detect should work out which modules have to be loaded to support whichever motherboard you are using. A chunk of the modules are currently shipped with the 2.2.17 kernel and the remainder are shipped with our lm_sensors and lm_sensors-drivers rpms in a catch as catch can fashion, generally if modules are built in both the kernel and the lm_sensors distribution I've gone with the kernel shipped version. However only the via686a set have been tested on anything. If you have problems.....

To get the stuff to load you'll need to do something like

Example 2. LCFG entry for via chipset

```
+update.modlist ANDALSO i2cdev sensors i2cviapro
+update.mod_i2cdev alias char-major-10-175 ic2-dev
```

/etc/sensors.conf sensors and sensord

Ok we have a bunch of modules which will dump raw data from the motherboard into /proc, however there is no naming convention between chipsets (or between motherboards) and no guarantee that what we're reading is scaled properly. So we have /etc/sensors.conf which allows you to manipulate the information coming out of sensors and to (re)set limits.

Unfortunately there is no convention over naming, input voltages can be tagged as in0, vin1, 2.0V etc. Worse there may be no coherent naming strategy for a specific chipset. e.g. the 686a chip tags voltages by level (2.0V) but the voltage min and max are tagged as in0_min etc. This doesn't help with writing a configuration object. As of this version obj-lm_sensor will work with via686a but I'm making no promises about other boards.

An example, Via 686 chispset w tyan S2390/Trinity KT motherboards

The via686 chipset on the tyan motherboard records the following information

- temp1: CPU temp (min,max).
- temp2: system temp (min,max).
- temp3: Sbr ??(min,max).
- fan1: CPU fan speed (min,dev).
- fan2: unused (could be additional system fan)(min,dev).
- 2.0V: CPU voltage (min,max).
- 2.5V: unknown & ignored (min,max).
- 3.3V: I/O voltage (min,max).
- 5.0V: unknown (min,max).
- 12.0V: unknown (min,max).

The voltage levels may not reflect the correct values and you may have to add a multiplier, the values I've set on the various Jupiter beowulf machines are taken from the lm_sensors mailing list.The lcfg entry looks like:

Example 3. LCFG entry for via chipset

```
lm_sensor.chip via686a-*
lm_sensor.ignore "2.5V"
lm_sensor.fans fan1 fan2
lm_sensor.fan_label_fan1 CPU FAN
lm_sensor.fan_min_fan1 3000
lm_sensor.fan_label_fan2 Sys FAN
lm_sensor.fan_min_fan2 0
lm_sensor.temps temp1 temp2 temp3
lm_sensor.temp_label_temp1 CPU Temp
lm_sensor.temp_over_temp1 50
lm_sensor.temp_hyst_temp1 45
lm_sensor.temp_label_temp2 Sys Temp
lm_sensor.temp_over_temp2 36
lm_sensor.temp_hyst_temp2 34
lm_sensor.temp_label_temp3 SBr Temp
lm_sensor.temp_over_temp3 26
lm_sensor.temp_hyst_temp3 24
lm_sensor.volts in0 in1 in2 in3 in4
lm_sensor.volt_label_in0 CPU Core
lm_sensor.volt_name_in0 2.0V
lm_sensor.volt_min_in0 1.5
lm_sensor.volt_max_in0 1.8
lm_sensor.volt_label_in1 IGNORE
lm_sensor.volt_name_in1 2.5V
lm_sensor.volt_min_in1 1.5
lm_sensor.volt_max_in1 1.8
lm_sensor.volt_label_in2 I/O
lm_sensor.volt_name_in2 3.3V
lm_sensor.volt_min_in2 3.3*0.90
lm_sensor.volt_max_in2 3.3*1.05
lm_sensor.volt_label_in3 +5V
lm_sensor.volt_name_in3 5.0V
lm_sensor.volt_min_in3 5.0*0.90
lm_sensor.volt_max_in3 5.0*1.10
lm_sensor.volt_label_in4 +12V
```

```
lm_sensor.volt_name_in4 12V
lm_sensor.volt_min_in4 12.0*0.95
lm_sensor.volt_max_in4 12.0*1.05
```

The first line defines the chipset being used, the second tells lm_sensors to ignore the voltage line marked "2.5V", the temps and fans are fairly straightforwards but the voltage stuff is a bit messy because of the lack of a naming convention. You may have to hack lm_sensors to get things to work. If you've got it all working properly then you should be able to do:

```
[callisto]iainr: sensors
via686a-isa-6000
Adapter: ISA adapter
Algorithm: ISA algorithm
CPU:       +1.72 V  (min =  +1.48 V, max =  +1.79 V)
I/O:       +3.22 V  (min =  +2.93 V, max =  +3.44 V)
+5V:       +4.77 V  (min =  +4.45 V, max =  +5.49 V)
+12V:     +11.51 V  (min = +11.39 V, max = +12.58 V)
CPU FAN:  4655 RPM  (min = 3000 RPM, div = 2)
Sys FAN:     0 RPM  (min =    0 RPM, div = 2)
CPU Temp: +44.2° C   (limit = +45° C, hysteresis = +50° C)
Sys Temp: +29.9° C   (limit = +34° C, hysteresis = +36° C)
SBr Temp: +24.8° C   (limit = +24° C, hysteresis = +26° C)

eeprom-i2c-0-51
Adapter: SMBus vt82c596 adapter at 5000
Algorithm: Non-I2C SMBus adapter
Memory type:            SDRAM DIMM SPD
SDRAM Size (MB):        256

eeprom-i2c-0-52
Adapter: SMBus vt82c596 adapter at 5000
Algorithm: Non-I2C SMBus adapter
Memory type:            SDRAM DIMM SPD
SDRAM Size (MB):        256
```

SNMP

disk

If *`disk filesystem min%|minspace`* is set then the snmp agent will report information about that filesystem via the .1.3.6.1.4.1.2021.9 (enterprises.ucdavis.dskTable.dskEntry) OID. If you define a minimum free space size (or percentage) and the free space falls below that then .1.3.6.1.4.1.2021.9.1.100.X will be set to 1 and an error message set in .1.3.6.1.4.1.2021.9.1.101.X (where X indicates the order of the *`disk`* definition).

So for the snmp object we can now define *`@disks filesys_$ size_$ percent_$`* if both percent and size are defined for a filesystem obj_snmp will default to percent.

pass

the *pass* directive allows the agent to hand off control of a branch of the OID tree to a script or program. The parameters are *OID executable* and *arguements*

```
pass .1.3.6.1.4.1.2021.255 /bin/sh /usr/local/sbin/cputemp
```

In LCFG this becomes.

```
+snmp.passes i2c
+snmp.oid_i2c .1.3.6.1.4.1.2021.255
+snmp.command_i2c /bin/sh /usr/local/sbin/cputemp
```

Once you have got this all up and running there are a number of tools which can be used to gether the information (snmpget/walk,tkined[7], rrdtool[8]) I've used NRG (largely becuase I've used it before).

Using NRG to gather stats

NRG[9] is a front end to rrdtool[10] which is a development of MRTG, essentially it's a script which uses snmpwalk and snmpget to pukll data via snmp and store it in an rrdtool database, the script then generates html and gif files to display graphs of the data. The script is configured using a metaconfiguration file and possibly a number of configuration files for specifig hosts.

To install add perl-Time-HiRes, rrdtool-1.0.33-2nrg and nrg-0.99.14-1dcs_nrg to your rpmcfg files, you'll also need to add something like

```
#NRG stuff
< Directory /usr/local/apache/web/nrg>
 Options ExecCGI
< /Directory>
AddHandler cgi-script .cgi
< Files "*.gif">
 ExpiresActive On
 ExpiresDefault M5
< /Files>
```

to /usr/local/apache/conf/www.conf in order to enable cgi and set the gifs to autoupdate. Secondly change NRG_WEB_TITLE in /usr/local/nrg/Makefile to reflect your web page. Finally you need to create NRG.mconf and .conf files, the NRG.mconf file looks something like

Example 4. NRG.mconf file.

```
#
# $Id: NRG.mconf.in,v 1.38 2001/04/12 14:34:34 rader Exp $
#

define(APACHE_D,/usr/local/nrg/bin/nrg-discover-apache -debug)
define(BIND_D,/usr/local/nrg/bin/nrg-discover-bind -debug)
define(ERROR_D,/usr/local/nrg/bin/nrg-discover-bind -debug)
define(IFACES_D,/usr/local/nrg/bin/nrg-discover-ifaces -debug)
define(PINGD_D,/usr/local/nrg/bin/nrg-discover-pingd -debug)
define(SENDMAIL_D,/usr/local/nrg/bin/nrg-discover-sendmail -debug)
define(SNMPD_D,/usr/local/nrg/bin/nrg-discover-snmpd -debug)
define(TABLE_D,/usr/local/nrg/bin/nrg-discover-tables -debug)
define(TCP_D,/usr/local/nrg/bin/nrg-discover-tcp -debug)
```

```
define(SOMESWITCH_IFACES,"SomeSwitch's Network Interface Data Table")
define(SOMESWITCH_ERRORS,"SomeSwitch's Network Interface Errors Data Table")
define(SITE_PING,"Jupiter's Ping Data Table")
define(SOMESERVER_TCP,"Jupiter's TCP Service Response Time Table")

# do NOT use trailing slash... it's tickles
# a nasty bug which hasn't been fixed yet...
WebRootDir[*]:       /usr/local/httpd/html
NRGSubDir[*]:        nrg
ConfFiles:           *.conf
BucketMconfTargets:  yes
HashBucketSize:      0
RunScript:           run-nrg

Directory:         /net/traffic
SomeSwitch:        IFACES_D public@1.2.3.4
SomeSwitch-iface:  TABLE_D -title SOMESWITCH_IFACES /net/traffic/SomeSwitch

Directory:         /net/errors
SomeSwitch:        ERROR_D public@1.2.3.4
SomeSwitch-err:    TABLE_D -title SOMESWITCH_ERRORS /net/errors/SomeSwitch

#Directory:        /apache
#SomeWebServer:    APACHE_D 1.2.3.5
```

Basically leave all the defines alone, for each host you want to gather info on define a directory to store the files in and the scripts you want to run to gather that info.

In the case of the lm_sensors details there are no autodiscovery scripts and for the moment we have to generate .conf files for each host. If you want the disk usage stats you can get these via SNMPD_D. Each .conf file looks like:

```
#
# callisto-temp.conf - graph temp stats for Callisto
#

# .*-s\d+$ matches *-s0, *-s1, ...

Variable[Callisto-temp][cputemp]:    enterprises.ucdavis.255.3 GAUGE
Variable[Callisto-temp][systemp]:    enterprises.ucdavis.255.4 GAUGE
Variable[Callisto-temp][othertemp]:    enterprises.ucdavis.255.5 GAUGE
Variable[Callisto-temp][fanspeed]:    enterprises.ucdavis.255.6 GAUGE
YLabel[Callisto-temp]:               Centigrade
Units[Callisto-temp]:                %sDegC
CalcDef[Callisto-temp][sfanspeed]:   fanspeed,200,/
Graph[Callisto-temp][cputemp]:       red LINE2
Graph[Callisto-temp][systemp]:       blue LINE2
Graph[Callisto-temp][othertemp]:     green LINE2
Graph[Callisto-temp][sfanspeed]:     black LINE2
Label[Callisto-temp][cputemp]:       "CPU temp"
Label[Callisto-temp][systemp]:      "System temp"
Label[Callisto-temp][othertemp]:      "Some other temp"
Label[Callisto-temp][sfanspeed]:      "Scaled fanspeed (rpm/200)"
LowerLimit[Callisto-temp]:           30
#---------------------------------------------

System[Callisto-temp]:            sommunity@callisto.dcs.ed.ac.uk
RRD[Callisto-temp]:               /jupiter/Callisto/callisto-temp.rrd
GraphWebPage[Callisto-temp]:      /jupiter/Callisto/callisto-temp.cgi

PageTitle[Callisto-temp]: Callisto temp data
PageTop[Callisto-temp]:   CPU and System temp data for Callisto
```

```
PageBody[Callisto-temp]:
  < TABLE>
    < TR> < TD>System:< /TD> < TD>Callisto CPU and system temps.< /TD>< /TR>
  < /TABLE>
< BR>
PageBottom[Callisto-temp]: Callisto CPU and system Temps.
```

It should be fairly clear what's going on, bear in mind that CalcDef uses reverse polish, more detailed documentation is at the NRG website. Once you've got the configuration files sorted out run **make rediscover** and nrg should go about building up the .conf files, do **make notify** to build required directories, rrd files and to build run-nrg which is the script which does the information gathering.

Finally set up cron to run run-nrg periodically (say every 5 minutes) and wait about 20 min or so for the .rrd files to fill up with info.

Notes

1. http://www.netroedge.com/~lm78/supported.html
2. http://www.csc.liv.ac.uk/~daves/Misc/UCD/guide.html
3. http://www.mrtg.org
4. http://nrg.hep.wisc.edu/
5. http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/
6. http://www2.lm-sensors.nu/~lm78/cvs/lm_sensors2/doc/chips/SUMMARY
7. http://wwwhome.cs.utwente.nl/~schoenw/scotty/
8. http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/
9. http://nrg.hep.wisc.edu/
10. http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/