Science & Technology Facilities Council

Technical Report

Monitoring scientific computing infrastructure using Nagios

Wadud Miah

January 2010

RAL-TR-2010-002

ISSN 1358-6254

RAL-TR-2010-002

Monitoring scientific computing infrastructure using Nagios

Wadud Miah[1]

This document will describe the Nagios configuration and use by the Scientific Computing Technology (SCT) group at the eScience department of STFC. The Nagios monitoring system has been extensively utilised for the purpose of increasing quality of service of the many services provided by the group. Clients of our services include software developers to scientists across Europe, so it is imperative that the high level of service that our clients expect is provided, and this is provided by a highly scalable, configurable and powerful monitoring system such as Nagios. The Nagios monitoring system is so widespread such that, in addition to standard host/service monitoring, grid monitoring tools such as NCG (Nagios Configuration Generator) are being developed and used and are phasing out existing monitoring such as Inca [3]. The fact that Nagios is an open source project makes it an even more attractive solution as users can modify the code to suit their needs and an example of this will be given. Due to the extensive use within academia and the open source community, knowledge of its use is widespread and support readily available on the World Wide Web.

At SCT, Nagios version 3.0.6 runs on RHEL 4.8 with Apache as the web server. This configuration works well, and is reported to run on many flavours of Linux [2]. This report will outline the experience and usage of Nagios within SCT and does not give an exhaustive overview of the technology; if readers require this, they are referred to [1].

[1] wadud.miah@stfc.ac.uk, Rutherford Appleton Laboratory

Scientific Computing Technology Group
eScience Department, Rutherford Appleton Laboratory
Oxfordshire, OX11 0QX

January 2010

Contents

1 Infrastructure

Nagios can be configured on a single or distributed system, where the latter provides redundancy and load distribution. The choice will largely depend on the number of checks - if a high number of checks are required, then a distributed set up is recommended; otherwise a standalone server will suffice. At SCT a distributed configuration is utilised due to the large number of hosts and services that are monitored. This involves a master server that hosts the web front end and three slave servers all of which are hosted on single dedicated physical hosts and not on virtual machines. The advantages on selecting physical hosts over virtual hosts are:

- The physical host will have the sole task of running Nagios checks, hence faster monitoring resulting in better quality of service;
- A requirement for paging alerting (see Section 5) is that the machine has access to a serial port which is connected to a modem. In virtual architectures, e.g. VMWare ESX(i) which is what is deployed at SCT, the technology does not provide access to a serial port;
- Faster response times for clients viewing service/host status on the web front end - one of the motivations of using a distributed configuration.

This distributed configuration is shown in Figure 1, where a modem is connected to the master server for the purpose of pager alerting. A modem is also connected to one of the slaves in the event that the master server suffers a fault, in which case an alert is sent and the recipient will then know that alerting is unavailable. The SLA to fix this fault has a higher priority than normal services, as part of the eScience department depends on the Nagios alerting system. This configuration provides a high level of monitoring in the event of failures.

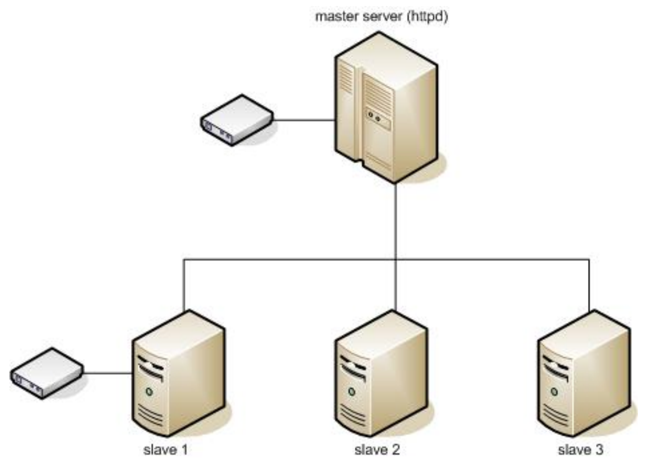

Figure 1: Distributed redundant Nagios infrastructure at SCT

The master server receives check results from the slaves and is configured to send out alerts by email or paging. The slaves execute the check and send the result to the master, and do not send alerts. In the event of a master server failure, web front ends also run on the slaves, but email or paging alerts will not be sent. However, the exception to this is slave 1 which sends out paging alerts in the event of a master server failure. During a slave failure, the master server seamlessly takes over checks that the slave is responsible for.

In Nagios, there are two types of checks: a host check and service check. The host check involves a simple ping using ICMP, whereas service checks are carried out using a number of Internet protocols. Host and service checks are described in Nagios as objects and provide a method of inheritance with the ability to over-ride characteristics of parent objects. This concept is well known in object oriented languages, and this functionality is provided in Nagios. In the distributed configuration, there are two types of parent host objects which are listed below:

Master Parent Host Object

```
define host {
  name                          hpcsg-host
  register                      0

  check_command                 check-host-alive
  max_check_attempts            3
  check_period                  24x7
  contact_groups                hpcsg
  notification_interval         0
  notification_period           24x7
  notification_options          d,u,r
  notifications_enabled         1
  obsess_over_host              0
}
```

Slave Parent Host Object

```
define host {
  name                          hpcsg-host
  register                      0

  check_command                 check-host-alive
  max_check_attempts            3
  normal_check_interval         30

  check_period                  24x7
  obsess_over_host              1
  notification_interval         0
  notification_period           none
  notification_options          n
  notifications_enabled         0
}
```

The directives to note are listed in Table 1.

Nagios directive	**Description of directive**
`register`	do not register this object; it is simply a parent
`obsess_over_host`	send results to master
`notification_options`	send notifications when host status changes
`notifications_enabled`	whether notifications should be sent

Table 1: Nagios parent host object directives

All host objects then inherit the characteristics of the above objects, using the `use hpcsg-host` directive. The two service parent objects are listed below:

Master Parent Service Object

```
define service {
  name                          hpcsg-service
  register                      0

  max_check_attempts            3
  normal_check_interval         30
  retry_check_interval          1
  active_checks_enabled         0
  passive_checks_enabled        1
  check_period                  24x7
  obsess_over_service           0
  notification_interval         0
  notification_period           24x7
  notification_options          w,u,c,r
  notifications_enabled         1
  contact_groups                hpcsg
}
```

Slave Parent Service Object

```
define service {
  name                          hpcsg-service
  register                      0

  max_check_attempts            3
  normal_check_interval         30
  retry_check_interval          1
  active_checks_enabled         0
  passive_checks_enabled        1
  check_period                  24x7
  obsess_over_service           1
  notification_interval         0
  notification_period           none
  notification_options          n
  notifications_enabled       0
}
```

The directives to note are listed in Table 2.

Nagios directive	Description of directive
`register`	do not register this object; it is simply a parent
`active_checks_enabled`	execute the check
`passive_checks_enabled`	await result of check
`obsess_over_service`	send result to master server
`notification_period`	period where notifications should be sent out
`notifications_enabled`	send notifications

Table 2: Nagios parent service object directives

Host and service checks at SCT are grouped together in host groups instead of service groups, and thus form a logical group. For example, a cluster and all its associated services are grouped together in the cluster name which is assigned a contact group. An example of this is shown below (for master and slave configurations):

Host Object Configuration

```
define host { # slave configuration
  use                         hpcsg-host

  host_name                   scarf.ac.uk
  alias                       scarf
  address                     1.2.3.4
  hostgroups                  SCARF
}

define host { # master configuration
  use                         hpcsg-host

  name                        scarf-host
  register                    0

  contact_groups              scarf
}

define host { # master configuration
  use                         scarf-host

  host_name                   scarf.ac.uk
  alias                       scarf
  address                     1.2.3.4
  hostgroups                  SCARF
}
```

Service Object Configuration

```
define service { # slave configuration
  use                   hpcsg-service

  hostgroup_name        SCARF
  service_description   disk-space
  check_command         check_nrpe!check_disk!10% 5% / /boot /tmp /var/log
  normal_check_interval 120
}

define service { # master configuration
  use                   hpcsg-service

  name                  scarf-service
  register              0

  contact_groups        scarf
  check_freshness       1
}

define service { # master configuration
  use                   scarf-service

  hostgroup_name        SCARF
  service_description   disk-space
  check_command         check_nrpe!check_disk!10% 5% / /boot /tmp /var/log
  freshness_threshold   7500
}
```

The practice adopted at SCT is to create parent object for each logical host group which is assigned a unique contact group. All objects that inherit that object will then have the same contact group, thus avoiding repetition. A notable directive for the slave service check is `normal_check_interval` (in minutes) which specifies the rate at which the check should be executed. The notable directives for the master service checks are `check_freshness` which specifies that the age of the check result be monitored, and `freshness_threshold` (in seconds) which specifies the time interval by which to consider the check result out of date and for the master to execute the check itself. The calculation used at SCT is:

$$\texttt{freshness\_threshold} = (\,\texttt{normal\_check\_interval} + \epsilon\,) \times 60 \tag{1}$$

where $\epsilon = 5$ and can be tweaked to obtain the optimal value. It is this mechanism that implements seamless redundancy.

2 Web front end and access control

Host and service check statuses can be viewed via a CGI web front end served over a secure connection using the UK eScience certificate authority [5]. Authentication is via UK eScience personal certificates. The Apache configuration for Nagios is given below:

```
ScriptAlias /nagios/cgi-bin /usr/local/nagios/sbin

<Directory /usr/local/nagios/sbin/>
  SSLVerifyClient Optional
  SSLVerifyDepth 3
  SSLCACertificateFile /etc/grid-security/certificates/adcbc9ef.0
  SSLOptions +FakeBasicAuth
  SSLUserName SSL_CLIENT_S_DN_X509
  SSLRequireSSL

  Options ExecCGI
  AllowOverride None
  AuthName "Nagios CGI Access"
  AuthType Basic
  AuthUserFile /etc/nagios/users-access
  Require valid-user
</Directory>

Alias /nagios /usr/local/nagios/share

<Directory /usr/local/nagios/share>
  SSLVerifyClient Optional
  SSLVerifyDepth 3
  SSLCACertificateFile /etc/grid-security/certificates/adcbc9ef.0
  SSLOptions +FakeBasicAuth
  SSLUserName SSL_CLIENT_S_DN_X509
  SSLRequireSSL

  Options None
  AllowOverride AuthConfig
  AuthName "Nagios CGI Access"
  AuthType Basic
  AuthUserFile /etc/nagios/users-access
  Require valid-user
</Directory>
```

The `AuthUserFile` contains certificate DNs of users who can have access to Nagios and an example is shown below:

```
/C=UK/O=eScience/OU=CLRC/L=RAL/CN=wadud miah:xxj31ZMTZzkVA
```

The `SSLOptions +FakeBasicAuth` directive specifies that the user need not provide a password when a valid certificate is passed to the web server, hence the hash of "password" is presented at the end of the DN. The directive `SSLUserName SSL_CLIENT_S_DN_X509` sets the client certificate DN as the user name.

In the file `/etc/nagios/cgi.cfg`, the directive `use_authentication=1` is required. However, as the user name is set to the X.509 certificate DN, the CGI cannot parse names with spaces, thus a mapping is required. This requires a modification (developed by SCT) of `cgi/cgiauth.c` and the code can be found in Section 9.1 of the Appendix. Once mapped to the user name, this can then be used to set access controls for the following directives in `/etc/nagios/cgi.cfg`:

```
authorized_for_system_information
authorized_for_configuration_information
authorized_for_system_commands
authorized_for_all_services
authorized_for_all_hosts
authorized_for_all_service_commands
authorized_for_all_host_commands
```

At SCT, the web front end is extensively used to manage host/service down time. This allows system administrators to get up to date information on what is in down time and why. This is achieved using the down time feature in the web front end and this is particularly useful for keeping other administrators up to date. During down time, Nagios suppresses alerting and this practice is highly encouraged as it enables proper management of a large number of hosts and services. If there are a large number of hosts/services that require down time, it can be time consuming and tedious to configure this for every single host/service via the CGI, and for this, a down time script has been developed to achieve this [4]. The syntax of the script is:

```
nagios_downtime.pl --mode <add|del>
                   --server <IP address of Nagios master server>
                   --hostname <name of host scheduled for downtime>
                   --service <name of service scheduled for downtime>
                   --downtime <downtime in minutes>
                   --comment <comment associated with downtime>
                   --user <user name to be used to access CGI>
                   --password <password to be used to access CGI>
```

To use this script, create a user specifically for this task and create a password using the `/usr/bin/htpasswd` command. Then add the user name and password to the `AuthUserFile` file, e.g.

```
nagios-downtime:dumbpasswordhash
```

Then add the user to `/etc/nagios/cgi.cfg`:

```
authorized_for_system_commands=nagios-downtime
authorized_for_all_service_commands=nagios-downtime
authorized_for_all_host_commands=nagios-downtime
```

This completes the process.

The Nagios web front end is access controlled to only allow SCT group members read/write access to the CGI. As the hosts/services monitored are of interest to external users, its status is provided to them via other web servers and this is depictured in Figure 2.

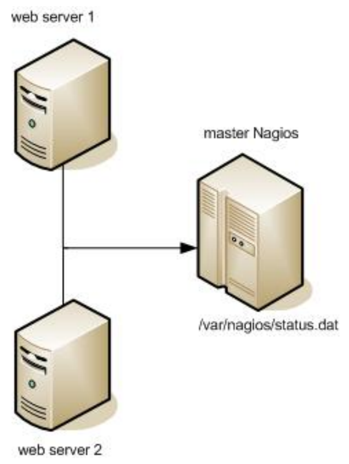

Figure 2: Host/service status provided to web servers by the master server

The web servers can obtain host/service status from the Nagios master server as this server contains the status of all hosts/services (see Figure 1). This configuration has the advantage of centralising host/service statuses which other monitoring tools can query. The master Nagios server contains a script which parses the Nagios log file[1] and is executed via NRPE. The script is given in Section 9.3 in the Appendix which prints the output of the check and exits with the exit code of the check itself.

3 Nagios checks

Nagios supports monitoring via a number of protocols which are listed below:

- Internet protocols, e.g. TCP, UDP, HTTP, ICMP;
- Secure shell, SSH;
- Nagios remote plugin executor, NRPE;
- Simple network monitoring protocol (based on UDP), SNMP;
- Nagios service check acceptor, NSCA.

The idea of monitoring is for checks to have minimal load on hosts, otherwise this will alter their state. Hence, simple methods are better suited for checks and for this reason, checks via the SSH protocol is not favoured at SCT. In addition to the load, this creates a huge burden on managing the large set of SSH keys. The protocols used at SCT are all listed above, save SSH, and the protocol most favoured is SNMP. Table 3 shows which protocols are used for the different monitoring requirements:

[1]specified by `log_file` in `/etc/nagios/nagios.cfg`

Internet protocols	**NRPE**	**SNMP**	**NSCA**
http, ssh, ntp	disk space, back up	RAID devices	SMART
ldap, dns	load, swap space	Panasas storage	software RAID
sql, smtp	processes, RAID	environmental sensors	SNMP traps
	memory, SMART		
	ECC, PSU		

Table 3: Nagios check protocol suitability

Internet protocols are mainly used to monitor standard network services. It is a quick check and has a minimal overhead and requires no configuration on the client, except local firewall rules, e.g. via TCP wrappers or IP tables.

NRPE allows Nagios to execute more complex plugins locally on a host. This method should be used when simple methods such as SNMP and checks based on the standard Internet protocols are not sufficient. Examples include determining the number of ECC errors a computational node has suffered and the SMART status of hard drives. This method allows Nagios to execute custom plugins which can be developed in C, C++, Java, shell script, etc, more of which will be mentioned in Section 4. NRPE consists of a daemon which runs as the `nagios` Linux user and listens typically on TCP port 5666. This is a lightweight daemon and has been reported to take up approximately 02:00 minutes of CPU time over a period of 5 months. This daemon is typically controlled by `/etc/nagios/nrpe.cfg`, and the practice at SCT includes:

- Listening on the management socket, i.e. <IP address>:5666, and is controlled by the `server_address` directive. This is to ensure that other devices, whether Ethernet or high performance interconnect (HPI), are not listening for NRPE packets;
- Only authorised Nagios servers are allowed to connect to this socket and this is controlled by `allowed_hosts` and IP tables;
- Allow commands to accept arguments by setting `dont_blame_nrpe` to unity;
- Building NRPE with the OpenSSL libraries to ensure secure connections. In a scientific computing environment where performance is preferred over security, it is recommended to use a low overhead encryption algorithm;
- Some commands, such as checking on RAID status, requires root privileges. As the checks are executed as the Linux `nagios` user, such checks are executed using the `sudo` mechanism. Although `sudo` can be configured to allow a low privileged user to execute plugins within a specified directory as root, the practice at SCT is instead to allow this user to execute only a set of specified check commands in the interest of security.

The SNMP Management Information Base (MIB) directory provides a whole array of information. For example, a RAID unit MIB will have information on hard drive, PSU, battery, logical drive status. The MIB is in text format and can be browsed to obtain useful metrics/values for the purpose of monitoring. As it is based on UDP (thus a low overhead query) and provides a mass of useful information it is the preferred protocol for the purpose of monitoring in Nagios at SCT.

The NSCA method is used in trap environments where a result is expected from a device. This is not so widely used within SCT as this creates a dependency on the device running normally, and in the event of device failure, Nagios will assume there are no problems. However, this can be mitigated by the `check_freshness` directive which determines if a check result is out of date, and if so, what action to take. This action is usually involves executing the plugin itself (if available). An example of this configuration at SCT is monitoring software RAID (mdadm), where a daemon runs in the background monitoring the arrays. In the event of a failure, it can be configured to execute a script which can feed the result to Nagios. If `check_freshness` deems the result out of date, the `mdadm` command can be executed to determine the status of the array. This has proven useful as a course check schedule can be configured as a level of trust is gained by the daemon, but not absolute, hence `check_freshness` usage. The NSCA mechanism can be used in more complex situations such as building monitoring systems that only provide SNMP trap functionality. In such circumstances, the Nagios status can not be entirely depended upon, but provides a simple counter measure for monitoring normal operations of IT infrastructure.

4 Plugins

Nagios provides a simple mechanism for administrators to develop their own plugins to extend its functionality. This is extremely useful as every infrastructure will have its unique characteristics, hence the importance of this requirement. Plugins are simply executables which print a few lines of output[2], and an exit code to determine the following levels of failure and success:

- Zero: the check executed successfully;
- One: the check raised a warning;
- Two: the check raised a critical alert and the host/service requires immediate attention;
- Three: result of check was unknown.

The development of plugins also falls under the software development life cycle with quality attributes. Considering that the purpose of monitoring is to determine the state of services provided with minimal impact on the host itself, it is imperative that the plugin is efficient with low emphasis on and at the expense of robustness. The impact on the host is important in high performance environments where CPUs are running at 100% utilisation 24 hours a day, 7 days a week. If checks are not lightweight and are executed frequently, they will take valuable CPU time in a) context switching b) generating interrupts, thus reducing the productivity and efficiency of the scientific computing infrastructure.

The plugin need not be robust in the sense of checking parameters passed to it, as these do not change often, if at all. As a quality control measure at SCT, when a check is implemented in Nagios, it is executed on the command line as well as via the Nagios web front end itself to ensure its correctness and robustness. For example, when checking the value of an Object Identifier (OID) in an MIB, there is little point in checking the OID which involves a branch:

```
if [ -z "$OID" ]
then
  echo "OID $OID does not exist"
```

[2]version 3.x of Nagios allows multi-line output, whereas 2.x only allows a single line of output

```
  exit 3
fi

VALUE=‘snmpget -v1 -c public -m IFT-SNMP-MIB 172.16.140.37 luDevStatus.8‘
```

This reduces the pipelining/efficiency of the check.

5 Alerting methods

The primary alerting mechanism provided by Nagios is email which is implemented as a command object with access to numerous macros (essentially shell variables). As alerting is via a command execution, other forms of alerting methods can be easily implemented. For High Business Impact (HBI) services, paging alerts are utilised at SCT. Paging was preferred over SMS alerts, even though mobile phones are ubiquitous, as paging provides a higher level of reliability:

- Pager signal coverage is more widespread than mobile phone signal. Messages can even reach pagers in basements;
- Do not suffer from mobile phone network overload;
- SMS traffic is generally considered low priority with no guarantee of receiving at a reasonable time, if received at all. Pager messages are sent with little delay.

The paging set up involves connecting a modem via the serial connection to the master Nagios server and a single slave server, as shown in Figure 1. The modems that work well and are deployed at SCT are a) MultiTech MultiModem ZBA and b) US Robotics 56K fax modem. The paging service provider is vodafone UK [7], which the modem dials and send messages using the Telelocator Alphanumeric Protocol (TAP). The software layer that implements TAP is Sendpage [6] which listens on the SNPP (TCP) port 444. An example Sendpage configuration file is given in Section 9.2.

Sendpage has other useful features as well as implementing TAP:

- Provides queue management which allows the queuing of multiple pager messages with a single modem dial;
- Error detection;
- Configures the Nagios server as a SNPP server. This allows clients to send pagers to it using the `/usr/bin/snpp` client tool [6].

The contact object contains the pager number of the recipient and an example is shown below:

```
define contact{
  contact_name                        miahw-pager
  alias                               Wadud pager
  host_notification_period            24x7
  service_notification_period         24x7
  service_notification_options          w,u,c,r
  host_notification_options             d,u,r
  service_notification_commands         service-notify-by-pager
  host_notification_commands            host-notify-by-pager
  pager                                 07659232191
}
```

where the two command objects are defined as:

```
define command{
  command_name    host-notify-by-pager
  command_line    /usr/bin/sudo /usr/bin/sendpage -f nagios@oberon
  -m "$HOSTNAME$ -> $HOSTSTATE$" $CONTACTPAGER$@oberon
}

define command{
  command_name    service-notify-by-pager
  command_line    /usr/bin/sudo /usr/bin/sendpage -f nagios@oberon
  -m "$HOSTNAME$: $SERVICEDESC$ -> $SERVICESTATE$" $CONTACTPAGER$@oberon
}
```

Note that the command line is on a single line. The macro `$CONTACTPAGER$` contains the pager number as defined in the contact. The above configuration will send alerts in the format:

```
HOST -> {UP,DOWN}
```

for host alerts and:

```
HOST: SERVICE-DESCRIPTION -> {UP,DOWN}
```

for service alerts. Additional information can be included in pager alerts using the many available macros [2].

6 Good practices

Deploying, maintaining and the management of Nagios, as with managing any other network service, requires the implementation of good practices to ensure the aims of monitoring are achieved, namely fast, reliable and resilient monitoring. This section will outline some of the good practices adopted at SCT.

6.1 Structuring configuration files

As mentioned in Section 1 host groups are used to group hosts/services that form a logical collection and host group view is frequently used in the web front. Following on from this, separate configuration files are used for each host group - a file for hosts within a host group and its respective services in a separate file. For example, for the SCARF cluster, the files that contain its configuration are `groups/scarf.cfg` and `groups/scarf-services.cfg`[3]. On the master server, the format of the files start with a comment which includes the file name and the slave server that is responsible for the active checks, e.g.

```
# -- /etc/nagios/groups/scarf.cfg -- slave1.rl.ac.uk
```

The slave server only includes the file name. When files are changed, a quick syntax check is provided by Nagios. This is recommended before restarting the Nagios Linux services:

```
[root@master nagios]# /usr/local/nagios/bin/nagios -v /etc/nagios/nagios.cfg
[ ... ]
Total Warnings: 0
Total Errors:   0

Things look okay - No serious problems were detected during the pre-flight check
[root@master nagios]# service nagios reload
```

6.2 Check frequencies

The frequency of the check is dependent on the type of check and the level of service. Type of check includes whether the check involves hardware (e.g. hard drive, memory) or software (web server) status; for hardware status a less frequent schedule is chosen as hardware is not expected to fail as often as software. If the service level agreement (SLA) is 6 hours and requires no more than hour to fix, then a check frequency could be every 5 hours. Table 4 lists the example checks and frequencies used at SCT.

Type of check	**Check frequency (minutes)**
Disk space	120
ssh	60
LSF	60
SMART	720
ECC	360
PSU	720
RAID	60
RAID controller	60
http	15

Table 4: Nagios check frequencies used at SCT

6.3 Plugin development

Nagios plugins are simply executables which print a meaningful message and return the appropriate exit code and can be developed in any of the compiled or scripting languages. As discussed in

[3]this is relative to the `/etc/nagios` directory

Section 4, it is important that the plugins are efficient and are lightweight and containing minimum branches. The preferred method of plugin development at SCT is using the C programming language which, of course, results in a compiled binary and is far more efficient than scripting. In addition, there are many C APIs available, e.g. LSF, which can be used to develop useful plugins.

An additional practice used at SCT is feeding data to Ganglia within the plugin. This is to prevent duplicate acquisition of the data for both Nagios and Ganglia and is another example of the efficient and lightweight monitoring approach. This example is shown below:

```
VAL=‘$SNMP_GET -v1 -m IT-WATCHDOGS-MIB -Cf -c public $IP_ADDR \
     $OID_NAME $OID_AVAIL $OID_VAR 2>/dev/null | awk ’{ print $NF }’‘

sudo gmetric --name=$GANGLIA_NAME --type=int16 --units=$UNIT \
             --spoof=172.16.140.157:sensor.rl.ac.uk --value=$VAL > /dev/null

if [ $? -ne 0 ]; then
  echo "UNKNOWN: could not gmetric $VAL"
  exit 3
fi

if [ $VAL -gt $CRITICAL ]; then
  echo "CRITICAL: value is greater than $CRITICAL"
  exit 2
elif [ $VAL -gt $WARNING]; then
  echo "WARNING: value is greater than $WARNING"
  exit 1
else
  echo "OK: value is less than $WARNING"
  exit 0
fi
```

Note that the status of the Ganglia metric command is checked; if this command fails, an unknown status is sent to Nagios. This is to ensure that data is correctly sent to Ganglia.

7 Machine room monitoring

The functionality of Nagios is extended to monitor temperature conditions of the machine room. Problems can occur if the computer room air conditioners (CRAC) fail and this has occurred in many machine rooms or data centres. When this occurs it is important to protect the significant investment in equipment from damage due to overheating and this is in the form of automated protection in Nagios. And for this, sensors have been placed on the front and the back of racks, as well as in front of air conditioning units which will detect failing (or increasing) temperatures quicker than the sensors on the racks. If the temperature conditions are deemed to be unsuitable the plugin forces the shut down of the servers/nodes and switches off the ports on PDUs. However, network switches and consoles are left powered on to allow the powering on of hosts/nodes once CRAC issues have been resolved.

The script that monitors the conditions of the machine room collates the state of the sensors that are already monitored by Nagios using the script in Section 9.3. This prevents the duplicate polling of the sensors and threshold configuration. The script can be either configured to simply alert or alert and force a shutdown and usage is:

```
sct-monitor.pl --room=<hpd|ups|hpd-east|hpd-west> \
               --critical=<critical percentage> \
               --warning=<warning percentage> \
               --unknown=<unknown percentage> \
               [--force] [--help] [--debug]
```

The script can monitor different machine rooms as specified by the `--room` parameter, and assigned different Nagios sensor checks for each of the rooms in the `$sensor` hash. The first element of the hash contains the Nagios sensor unit host name followed by the Nagios sensor service check names. The switch `--force` actually shuts down the machine room and the script can be found in Section 9.4. The criteria for a shut down are:

```
$critical_percentage >= $shutdown_critical AND
$warning_percentage <= $shutdown_warning AND
$unknown_percentage <= $shutdown_unknown
```

where the `$critical_percentage` is calculated from the number of Nagios sensor checks in the critical state, and likewise for `$warning_percentage` and `$unknown_percentage`. The variables on the right hand side of the inequality are passed as parameters to the script. For each machine room monitored, there are two checks:

- `room-in-hour` which simply alerts and does not shut down the machine room and is executed during business hours. This check ommits the `--force` switch;
- `room-out-of-hour` which alerts and shuts down the machine room and out of office hours. This check uses the `--force` switch.

The schedules are defined using the `timeperiod` Nagios object.

The script in Section 9.3 calls an additional script for shutting down hosts/nodes. For each of the rooms, the `@hosts` array contains the hosts to be shut down and are read from data files and `$pdu` hash contains the details of the PDUs and the ports. The first element of the hash contains the private IP address of the PDU followed by the ports that are required *not* to be switched off. All other PDU ports are switched off. The shut down command is executed via SSH, so the Nagios user's public key must be distributed to all hosts/nodes. The command objects for monitoring the machine room are then given as:

```
define command{
  command_name  sct-monitor
  command_line  /usr/local/nagios/libexec/sct-monitor.pl --room=$ARG1$
                --critical=$ARG2$ --warning=$ARG3$ --unknown=$ARG4$ --debug
}
```

```
define command{
  command_name  sct-monitor-force
  command_line  /usr/local/nagios/libexec/sct-monitor.pl --room=$ARG1$
                --critical=$ARG2$ --warning=$ARG3$ --unknown=$ARG4$
                --force --debug
}
```

Note the command line is on a single line. The second object forces the shut down of hosts/nodes.

8 Summary

To ensure a good level of quality of service, it is imperative that service providers have a good idea on the status of their services within a reasonable time frame, and this must be before their users are negatively impacted by service degradation. An example of pre-emptive action taken from a Nagios alert is host/web certificate expiry ensuring it is renewed before expiry. Users of services expected a high quality of service availability, and understandably expect near 100% uptime. This is understood by the SCT group and Nagios has played a major role in meeting this level of service. Processes are in place to deal with service degredation/downtime after Nagios has notified the group of its status. Nagios has proven its ability in monitoring the complex computational infrastructure that the SCT group is responsible for by providing a number ways of monitoring, most of which are based on standard Internet protocols (see Table 3). This not only provides familiarity of the protocols, but more importantly the stability that is required in monitoring. There is no doubt that Nagios has resulted in greater up time, greater quality of service and better problem resolution.

8.1 Future plans

The Nagios monitoring system is also categorised as a service and falls under the service lifecycle. As the SCT infrastructure is being expanded and new services are being offered, further checks are required to ensure this continuous quality of service. Thus, the following is a list of future Nagios usage for the SCT group:

1. Grid monitoring using Nagios Configuration Generator (NCG). This project is in the development phase;
2. Monitoring machine room conditions such as a) computer room air conditioner (CRAC) unit status b) uninterruptable power supply (UPS) status;
3. Monitoring host/node status via intelligent platform management interface (IPMI). This has been recently configured due to the instability/complexities in earlier versions of IPMI;
4. Water sensors placed on SCT racks and monitor its status. This is in the event of a leakage from ceilings and when water cooled racks are installed.

Essentially, the aim of the SCT group is to consolidate and centralise all monitoring in Nagios for the eScience department, whether the service is in test or production. And if in test phase, this can also form part of the acceptance process. If other systems need access to host/service statuses then they can query Nagios rather than introduce additional parallel monitoring systems.

9 Appendix

9.1 Modification to map certificate DN to user name

The following modification is made to `cgi/cgiauth.c` to allow mapping of certificate DN to a unique user name.

```
const char DELIM = ':';

int read_data(FILE *fp, char *dn, char *user) {
  char c;
  int i;

  i=0;
  while ( (c = fgetc(fp)) != EOF && c != DELIM ) {
    dn[i++] = c;
  }

  dn[i]='\0';

  i=0;
  while ( (c = fgetc(fp)) != EOF && c != '\n' ) {
    user[i++] = c;
  }

  user[i]='\0';

  return ( c == EOF ) ? EOF : 1;
}

int get_authentication_information(authdata *authinfo){
        mmapfile *thefile;
        char *input=NULL;
        char *temp_ptr;
        int needed_options;

        char dn[200], user[200] = "nobody";
        const char *file = "/etc/nagios/dn-map";
        FILE *fp = fopen(file,"r");

/*.. more code ..*/

        temp_ptr=getenv("REMOTE_USER");
        if(temp_ptr==NULL){
                authinfo->username="";
                authinfo->authenticated=FALSE;
                }
        else{
               authinfo->username=(char *)malloc(strlen(temp_ptr)+1);
               if(authinfo->username==NULL)
                       authinfo->username="";
               else

        /* map DN to username here and store result in temp_ptr
              this will then be copied to authinfo->username */

                  if ( fp == NULL ) {
                    return ERROR;
                  }

                  while ( read_data(fp,dn,user) != EOF ) {
                    if ( strcmp(temp_ptr,dn) == 0 ) {
                      strcpy(authinfo->username,user);
                      temp_ptr=user;
                    }
                  }

/*.. more code ..*/
```

The syntax of /etc/nagios/dn-map is <X.509 certificate DN>:<user name>.

9.2 Example Sendpage configuration file

Below is an example /etc/sendpage.cf configuration file:

```
##############################################################################
#                                 sendpage.cf                                #
##############################################################################
#
# There are four majors sections:
#       - global        Any global settings
#       - "modem"       Each modem's settings
#       - "pc"          Each Paging Central's settings
#       - "recip"       Each recipient name's settings
#
# Except for global, each section starts with the section name in
# []'s.  So, to define a modem named "sportster", the section name
# would be "[modem:sportster]", and all of the sportster's settings
# would follow.
#
# section names cannot have "=", "@", or ":" in their text.
#

#############################
# global section
#############################
```

```
# queue & manager-level debugging.  Default is "false"
#
debug = true

# select-loop debugging.  Default is "false"
#       Leave this as it is unless you're digging around in the select
#       loop code.  It is VERY annoying.  :)
#

# SNPP activity debugging.  Default is "false"
#
debug-snpp = true

# alias-expansion debugging.  Default is "false"
#
alias-debug = false

# Filename prefix for writing process ID files.
#       Default is "/var/spool/sendpage/sendpage"
#
pidfileprefix = /var/spool/sendpage/sendpage

# Filename prefix for writing UUCP-style device locks.
#       Default is "/var/lock/LCK.."
#
lockprefix = /var/lock/LCK

# Directory to store Paging Central pager queues
#       Default is "/var/spool/sendpage"
#
queuedir = /var/spool/sendpage

# Username that sendpage should be running as.
#       Default is "sendpage"
user = sendpage

# Group sendpage needs to lock devices.
#       Default is "uucp"
group-lock = lock

# Group sendpage needs to read/write devices.
#       Default is "tty"
group-tty = uucp

# Email address that page emails claims to be coming from.
#       Default is "sendpage"
#
page-daemon = nagios@oberon
```

```
# Will page-daemon be Cc'd on email failures?
#       Default is "true"
cc-on-error = false

# By which mechanism should email be delivered?  mail, sendmail, or SMTP?
#       Default is "sendmail"
mail-agent = sendmail

# Should page senders be notified about permanent failures?
#       Default is "true"
fail-notify = true

# Page senders should be notified every Xth temporary failure.
# (0 means 'never')
#       Default is "5".
tempfail-notify-after = 5

# How many times does a page hit a temporary error before failing forever?
#       Default is "20"
max-tempfail = 20

# Should syslog be used instead of STDERR for logging?
#       Default is "true"
syslog = true

# When using syslog, which syslog options should be used?
# (any of "pid", "ndelay", "cons", or "nowait")
#       Default is "pid"
syslog-opt = pid

# Which syslog facility should be used?  man syslog for more info
#       Default is "daemon"
syslog-facility = daemon

# What port sendpage binds to for the SNPP server.
#       Default is "444"
snpp-port = 444

# What local address sendpage binds the SNPP server to.
#   *NOTE*
#       You should set this to "0.0.0.0" if you want to receive pages from
#       the rest of the world.
#
#       You can use this to limit which IP address SNPP is bound to.
#   *NOTE*
#
#       Default is "localhost"
snpp-addr = 0.0.0.0
```

```
# What to do about incoming SNPP requests (multiple entries allowed)
#   *NOTE*
#       You should set this to "0.0.0.0/0.0.0.0:ALLOW" if you want
#       anyone to connect to your SNPP server.
#   *NOTE*
#
#       Format is "NET/MASK:WAY" where WAY is either "ALLOW" or "DENY"
#
#       ACL processing is done top to bottom, and if no match occurs,
#       the connection is rejected.
#
#       Default is "127.0.0.1/255.255.255.255:ALLOW"

#snpp-acl="128.174.5.0/255.255.255.0:ALLOW"
#snpp-acl="128.23.1.10/255.255.255.255:DENY"
#snpp-acl="128.23.1.0/255.255.255.0:ALLOW"

# Command to run after each successful or failed page
#       Default is unset
#       Command gets contents of page on stdin, and 2 command line
#       parameters:
#               arg 1: status (0=page failed, 1=page succeeded)
#               arg 2: page alias (who was paged)
#
#completion-cmd = "/usr/local/bin/page-sent"

######################
# modem configuration
#       Each section should be called "modem".  (e.g.  "[modem:sportster]")
######################

# My first "modem" section.  I named it "sportster" because that's what it is
[modem:multitech]

# Should this modem's character-level debugging be turned on?
#       Default is "false"
debug   = false

# This modem's transmission settings.
#       Defaults are data=7, parity=even, stop=1, flow=rts,
#                    baud=9600, strict-parity=false
data           = 8
parity         = none
stop           = 1
flow           = rts
baud           = 9600
strict-parity  = false

# Which device this modem should use
#       Default is "/dev/null", so you better specify one.  :)
dev     = /dev/ttyS0

# This modem's initialization string
#       Default is "ATZ"
init   = "ATZ"

# This modem's "okay" response string (this is a regexp)
#       Default is "OK"
initok = "OK"

# What to look for if something has gone wrong while init'ing (this is a regexp)
#       Default is "ERROR"
error = "ERROR"

# How many seconds to wait for initok after init with this modem
#       Default is "4"
initwait = 8

# How many times to try to initialize the modem
#       Default is "2"
initretries = 4

# The dialing prefix for this modem
#       Default is "ATDT"
dial = "ATD"

# The telephone prefix to get a dialtone out of the building (for PBXs, etc)
#       Default is ""
dialout = ""

# The areacode this modem has (for figuring areacode matches with PCs)
#       Default is unset
# If you never use area code, either make this "-" or don't use "areacode"
# options in the PC definitions.
areacode = "-"

# What to look for after connecting successfully (this is a regexp)
#       Default is "CONNECT.*\r"

dialok = "CONNECT.*\r"

# What to look for if something goes wrong while dialing (this is a regexp)
#       Default is "ERROR|NO CARRIER|BUSY|NO DIAL|VOICE"
no-carrier = "ERROR|NO CARRIER|BUSY|NO DIAL|VOICE"

# How many seconds to wait for dialing to connect
#       Default is "60"
dialwait = 60
```

```
# How many times to try and redial (unimplemented, actually...)
#       Default is "3"
dialretries = 3

# How should "carrier detection" be done?  "on", "off", "dsr"
# "DSR" can be used when a cable or OS doesn't correctly provide CD
#       Default is "on"
carrier-detect = on

# How many seconds should the DTR be held down during initialization?
#       Default is "0.5"
dtrtime = 0.5

#########################
# Paging central section
#       each section should be called "pc" (e.g. "[pc:ameritech]")
#########################
[pc:oberon]
# Is this PC enabled?  Set to false to stop processing a PC, for example
#       Default is "true"
enabled = true

# This PC's protocol-level debugging.
#       Default is "false"
debug   = false

# Email address that page emails claims to be coming from.
#       Default is unset, and will fall back to the global "page-daemon" setting
#
page-daemon = nagios@oberon

# Will page-daemon be Cc'd on email failures for this PC?
#       Default is unset, and will fall back to global "cc-on-error" setting
#cc-on-error = true

# Should page senders be notified about permanent failures?
#       Default is unset; will fall back to global option
# fail-notify = true

# Page senders should be notified every Xth temporary failure.
# (0 means 'never')
#       Default is unset; will fall back to global option
tempfail-notify-after = 10

# How many times does a page hit a temporary error before failing forever?
#       Default is unset; will fall back to global option
max-tempfail = 10
```

```
# vodafone pager server number
phonenum = 07699121314

# How many pages can be sent in each session with this PC?
#       Default is 0 (unlimited)
maxpages = 0

# How many blocks can be sent in each session with this PC?
#       Default is 0 (unlimited)
maxblocks = 0

# How many characters can be sent in each page for this PC? (For UCP, not TAP)
#       Default is 1024
maxchars = 1024

# How many characters per block are allowed during TAP transmission?
# The protocol normally has this at "250" (due to the 256 limit, and
# encoding requires 6 chars).  Making this higher than 250 isn't sensible,
# but some TAPs need it smaller.
#       Default is "250"
chars-per-block = 250

# How many times are we allowed to split up a page that exceeds the
# max chars limit?  (For example, if maxchars was "100" and maxsplits was
# "5" and someone sent a 2000 character page, sendpage would generate
# five 100-character pages before cutting off the page.)
#       Default is "6"
maxsplits = 6

# Which TAP protocol to use.  Should be one of "PG1", "PG3", or "UCP"
#   Regular TAP PagingCentrals are "PG1".
#   UCP PagingCentrals will need "UCP".
# If you had a "pet3" style PC before, this needs to be "PG3"
#       Default is "PG1"
proto = PG1

# How many fields does the PC expect to be getting during Block Transmission?
# If you had a "pet3" style PC before, this needs to be "3".
#       Default is "2"
fields = 2

# What is the password for accessing this Paging Central?
#       Default is "000000".  Shouldn't be more than 6 characters.
#password=123456

# Should we assume strict TAP protocol, and require CR before each answer?
# If you can set this to true, do so, as it makes textual response codes
# easier to read.  However, very few PCs use those codes, and very few
# PCs have correctly implemented strict TAP, so it's unlikely you want this.
#       Default is "false"
stricttap = false

# Characters less than 0x20 are allowed in a block's field?
# If you can set this, it makes pages prettier (can send tabs, newlines,
# etc), but some PCs really don't like this.  See 'esc' and 'lfok' options.
#       Default is "false"
ctrl = false

# Can characters less than 0x20 be escaped, as in TAP spec 1.8?
# If you can't set "ctrl" to true, see if this one set to true works.
#       Default is "false"
esc = false

# Is LF explicitly allowed by this PC? (only useful if "ctrl=false")
#       Default is "false"
lfok = false

# Can fields be split across blocks?
#       Default is "true"
fieldsplits = true

# How many seconds to wait before sending CR when waiting for the ID= tag?
#       Default is "2", from the T1 of the TAP protocol
answerwait = 2

# How many retries to allow before giving up waiting for the ID= tag?
#       Default is "3", from the N3 of the TAP protocol
answerretries = 3

# How many seconds before we giving up trying to dial this PC?
#       Default is whatever the modem's dialwait is
#dialwait=20

# How many seconds should this PC wait between queue scans?
#       Default is "20"
rundelay = 20
```

9.3 Script to obtain host/service status

The script below parses the log file `/var/nagios/status.dat` to obtain the status of a service associated with a host.

```
#!/usr/bin/perl -w

use Getopt::Long;

my $nagios_status = '/var/nagios/status.dat';
my ( $host, $service );
my $found = 0;

GetOptions( "host=s" => \$host, "service=s" => \$service ) or
  die "Could not process arguments\n";

if ( (!defined $host) || (!defined $service) ) {
  usage( 1 );
}

open( my $status, $nagios_status ) or
  die "Could not open file $nagios_status $!\n";

while ( my $line = <$status> ) {
  chomp( $line );

  if ( $line =~ /^service/ ) {
    while ( my $directive = <$status> ) {
      chomp( $directive );

      if ( $directive =~ /^\s+host_name/ ) {
        my $host_name = (split( /=/, $directive ))[1];

        if ( $host =~ /^$host_name$/ ) {
          chomp( $directive = <$status> );

          my $service_name = (split( /=/, $directive ))[1];

          if ( $service_name =~ /^$service$/ ) {
            my ( $current_state, $plugin_output );
              $found = 1;

              while ( $directive = <$status> ) {
                chomp( $directive );

                if ( $directive =~ /^\s+current_state=(.*)/ ) {
                  $current_state = $1;
                } elsif ( $directive =~ /^\s+plugin_output=(.*)/ ) {
                  $plugin_output = $1;
                }

                if ( (defined $current_state) && (defined $plugin_output) ) {
                  close( $status ) or
                    die "Could not close file $nagios_status $!\n";

                    print "$plugin_output\n";

                    exit $current_state;
                }
              }
            }
          }
        } elsif ( $directive =~ /^\s+}/ ) {
          last;
        }
      }
    }
  }

unless ( $found ) {
  print "Error: could not find host $host *and* service $service\n";
  close( $status ) or
    die "Could not close file $nagios_status $!\n";

  exit 2;
}

sub usage {
  my $rc = shift;

  print "Usage: nagios-check.pl --host <host name> --service <service name>\n";

  exit $rc;
}
```

9.4 Script to monitor machine room conditions

```
#!/usr/bin/perl -w

use Getopt::Long;
use constant NAGIOS_UNKNOWN   => 3;
use constant NAGIOS_CRITICAL  => 2;
use constant NAGIOS_WARNING   => 1;
use constant NAGIOS_SUCCESS   => 0;
use constant CHECK_SERVICE    =>
    '/usr/local/nagios/libexec/nagios_service_poll';
use constant SHUTDOWN_SCRIPT  => '/usr/nagios/sct-shutdown.pl';
use constant SHUTDOWN_LOG     => '/usr/nagios/log/sct-monitor';
use strict;

my ( $room, $critical, $warning, $success, $unknown, $rc ) =
   ( 0, 0, 0, 0, 0, 0 );
my ( $shutdown_critical, $shutdown_warning, $shutdown_unknown ) =
   ( -1, -1, -1 );
my ( $total_checks, $help, $force, $debug ) = ( 0, 0, 0, 0 );
my ( $critical_percentage, $warning_percentage, $success_percentage,
     $unknown_percentage, $command );
```

```
my $sensors = {};

GetOptions( "room=s" => \$room,
            "critical=i" => \$shutdown_critical,
            "warning=i" => \$shutdown_warning,
            "unknown=i" => \$shutdown_unknown,
            "force" => \$force, "help" => \$help,
            "debug" => \$debug )
  or usage( 1 );

usage( 0 ) if ( $help );
usage( 1 ) if ( $shutdown_critical == -1 ||
                $shutdown_warning == -1 ||
                $shutdown_unknown == -1 );

if ( $room =~ /^HPD$/i ) {
  $sensors->{SENSOR_01}->[0] = 'sensor1'; # name of nagios host
  $sensors->{SENSOR_01}->[1] = 'scarf06-back-1'; # name of nagios service check
  $sensors->{SENSOR_01}->[2] = 'scarf06-back-2';
  $sensors->{SENSOR_01}->[3] = 'scarf06-back-3';
  $sensors->{SENSOR_01}->[4] = 'scarf06-front-1';
  $sensors->{SENSOR_01}->[5] = 'scarf06-front-2';
  $sensors->{SENSOR_01}->[6] = 'scarf06-front-3';

  $sensors->{SENSOR_02}->[0] = 'sensor2';
  $sensors->{SENSOR_02}->[1] = 'ngs-back-1';
  $sensors->{SENSOR_02}->[2] = 'ngs-back-2';
  $sensors->{SENSOR_02}->[3] = 'ngs-back-3';
  $sensors->{SENSOR_02}->[4] = 'ngs-front-1';
  $sensors->{SENSOR_02}->[5] = 'ngs-front-2';
  $sensors->{SENSOR_02}->[6] = 'ngs-front-3';

  # any other host and sensor service check
} elsif ( $room =~ /^UPS$/i ) {
  $sensors->{SENSOR_10}->[0] = 'sensor10';
  $sensors->{SENSOR_10}->[1] = 'sct-ups-back-1';
  $sensors->{SENSOR_10}->[2] = 'sct-ups-front-1';

} elsif ( $room =~ /^HPD-EAST$/i ) {
  $sensors->{SENSOR_01}->[0] = 'sensor1';
  $sensors->{SENSOR_01}->[1] = 'scarf06-back-1';
  $sensors->{SENSOR_01}->[2] = 'scarf06-back-2';
  $sensors->{SENSOR_01}->[3] = 'scarf06-back-3';
  $sensors->{SENSOR_01}->[4] = 'scarf06-front-1';
  $sensors->{SENSOR_01}->[5] = 'scarf06-front-2';
  $sensors->{SENSOR_01}->[6] = 'scarf06-front-3';

  $sensors->{SENSOR_04}->[0] = 'sensor4';
  $sensors->{SENSOR_04}->[1] = 'scarf08-back-1';
  $sensors->{SENSOR_04}->[2] = 'scarf08-back-2';
  $sensors->{SENSOR_04}->[3] = 'scarf08-front-1';
  $sensors->{SENSOR_04}->[4] = 'scarf08-front-2';

} elsif ( $room =~ /^HPD-WEST$/i ) {
  $sensors->{SENSOR_02}->[0] = 'sensor2';
  $sensors->{SENSOR_02}->[1] = 'ngs-back-1';
  $sensors->{SENSOR_02}->[2] = 'ngs-back-2';
  $sensors->{SENSOR_02}->[3] = 'ngs-back-3';
  $sensors->{SENSOR_02}->[4] = 'ngs-front-1';
  $sensors->{SENSOR_02}->[5] = 'ngs-front-2';
  $sensors->{SENSOR_02}->[6] = 'ngs-front-3';

  $sensors->{SENSOR_03}->[0] = 'sensor3';
  $sensors->{SENSOR_03}->[1] = 'sctgp-back-1';
  $sensors->{SENSOR_03}->[2] = 'sctgp-back-2';
  $sensors->{SENSOR_03}->[3] = 'sctgp-front-1';
  $sensors->{SENSOR_03}->[4] = 'sctgp-front-2';
} else {
  usage( 1 );
}

my $log_file = SHUTDOWN_LOG . "-$room" . '.log';

open( my $log, '>', $log_file ) or
  die "could not open log file " . $log_file . " $!\n";

my $now = scalar( localtime time );
print $log $now . "\n";

foreach my $sensor ( keys %{$sensors} ) {
  my @sensor_detail = @{$sensors->{$sensor}};
  my $unit_name = $sensor_detail[0];
  my @sensor_names = @sensor_detail[1..$#sensor_detail];

  foreach my $name ( @sensor_names ) {
    $command = CHECK_SERVICE . " --host=$unit_name --service=" .
               $name . " > /dev/null ";
    $rc = system( $command );
    $rc = $rc >> 8;

    print $log "service check command = $command; rc = $rc\n";

    if ( $rc == NAGIOS_CRITICAL ) {
      $critical++;
    } elsif ( $rc == NAGIOS_WARNING ) {
      $warning++;
    } elsif ( $rc == NAGIOS_SUCCESS ) {
      $success++;
    } else {
      $unknown++;
    }

    $total_checks++;
  }
}

$success_percentage  = ( $success / $total_checks ) * 100;
$warning_percentage  = ( $warning / $total_checks ) * 100;
$critical_percentage = ( $critical / $total_checks ) * 100;
$unknown_percentage  = ( $unknown / $total_checks ) * 100;

if ( $critical_percentage >= $shutdown_critical &&
     $warning_percentage <= $shutdown_warning &&
     $unknown_percentage <= $shutdown_unknown ) {
  if ( $force ) {
    print "Critical: $room has $critical_percentage% sensor failure.
           SHUTTING DOWN\n";
    print $log "Critical: $room has $critical_percentage% sensor failure.
                SHUTTING DOWN\n";
    $command = SHUTDOWN_SCRIPT .
               " --room=$room --debug --sleep --force --pdu >>
                /usr/local/nagios/shutdown/log/ssh-out.dat 2>&1";

    print $log $command . "\n" if ( $debug );
    $rc = system( $command );
    $rc = $rc >> 8;

    if ( $rc != NAGIOS_SUCCESS ) {
      print "Critical: could not execute the shutdown command. run the
             command $command --debug!!\n";
      print $log "Error: could not execute the shutdown command. run the
                  command $command --debug!!\n";

      close( $log ) or
        die "could not close file $log_file $!\n";

      exit( NAGIOS_CRITICAL );
    }
  } else {
    print "$room has $critical_percentage% sensor failure.
           not shutting down\n";
    print $log "$room has $critical_percentage% sensor failure.
                not shutting down\n";
  }

  close( $log ) or
    die "could not close file $log_file $!\n";
  exit( NAGIOS_CRITICAL );
} elsif ( $warning_percentage > $shutdown_warning ) {
  print "Warning: warning percentage $warning_percentage is higher
         than $shutdown_warning\n";
  print $log "Warning: warning percentage $warning_percentage is
              higher than $shutdown_warning\n";

  close( $log ) or
    die "could not close file $log_file $!\n";

  exit( NAGIOS_WARNING );
} elsif ( $unknown_percentage > $shutdown_unknown ) {
  print "Unknown: unknown percentage $unknown_percentage is higher
         than $shutdown_unknown\n";
  print $log "Unknown: unknown percentage $unknown_percentage is
              higher than $shutdown_unknown\n";

  close( $log ) or
    die "could not close file $log_file $!\n";

  exit( NAGIOS_UNKNOWN );
} else {
  print "OK: $room has $critical_percentage% sensor failure.
         not shutting down\n";
  print $log "OK: $room has $critical_percentage% sensor failure.
              not shutting down\n";

  close( $log ) or
    die "could not close file $log_file $!\n";

  exit( NAGIOS_SUCCESS );
}

sub usage {
  my $rc = shift;

  print <<EOF;
Syntax: $0 --room=<hpd|ups|hpd-east|hpd-west>
           --critical=<critical percentage>
           --warning=<warning percentage>
           --unknown=<unknown percentage>
           [--force] [--help] [--debug]
EOF

  exit( $rc );
}
```

9.5 Script to shut down machine room

```
#!/usr/bin/perl -w

use Getopt::Long;
use constant SHUTDOWN_LOG => '/usr/nagios/log/sct-shutdown';
use strict;

my $passwd_file = '/usr/nagios/snmp_community.dat';
my @all_ports = ( 1..24 );
my @hosts = ();
my $pdus = {};
my ( $room, $sleep, $force, $rc, $debug, $help, $pdu, $command ) =
   ( 0, 0, 0, 0, 0, 0, 0, 0 );

GetOptions( "room=s" => \$room, "sleep" => \$sleep,
            "force" => \$force, "debug" => \$debug,
            "help" => \$help, "pdu" => \$pdu )
              or usage( 1 );
usage( 0 ) if ( $help );

if ( $room =~ /^HPD$/i ) { # shutdown entire HPD room
  @hosts = qw( scarf-hpd.dat ngs-hpd.dat nsccs-hpd.dat
               general-hpd.dat esx-hpd.dat service-hpd.dat
               panfs-hpd.dat );

  # set the PDU hash
  $pdus->{SCARF_PDU01}->[0] = '172.16.1.2';
  $pdus->{SCARF_PDU01}->[1] = 16;

  $pdus->{SCARF_PDU02}->[0] = '172.16.1.3';
  $pdus->{SCARF_PDU02}->[1] = 2;
  $pdus->{SCARF_PDU02}->[2] = 15;

  # any other PDUs and ports
} elsif ( $room =~ /^UPS$/i ) { # shutdown hosts in the UPS room
  @hosts = qw( general-ups.dat esx-ups.dat service-ups.dat );

  $pdus->{GENERAL_PDU16}->[0] = '172.16.1.4';
  $pdus->{GENERAL_PDU16}->[1] = 1;

  $pdus->{GENERAL_PDU17}->[0] = '172.16.1.5';
  $pdus->{GENERAL_PDU17}->[1] = 9;
} elsif ( $room =~ /^HPD-EAST$/i ) {
  @hosts = qw( scarf-hpd.dat );

  # set the PDU hash
  $pdus->{SCARF_PDU01}->[0] = '172.16.1.6';
  $pdus->{SCARF_PDU01}->[1] = 15;

  $pdus->{SCARF_PDU02}->[0] = '172.16.1.7';
  $pdus->{SCARF_PDU02}->[1] = 2;
} elsif ( $room =~ /^HPD-WEST$/i ) {
  @hosts = qw( ngs-hpd.dat nsccs-hpd.dat general-hpd.dat
               esx-hpd.dat service-hpd.dat );

  $pdus->{NGS_PDU01}->[0] = '172.16.1.8';
  $pdus->{NGS_PDU01}->[1] = 17;

  $pdus->{NGS_PDU02}->[0] = '172.16.1.9';
  $pdus->{NGS_PDU02}->[1] = 17;
} else {
  usage( 1 );
}

my $log_file = SHUTDOWN_LOG . "-$room" . '.log';

open( my $log, '>', $log_file ) or
  die "could not open file $log_file $!\n";

my $now = scalar( localtime time );
print $log $now . "\n";

foreach my $file ( @hosts ) {
  my $filename = "hosts/" . $file;

  open( my $fh, $filename ) or
    die "Could not open file $filename $!\n";
  # iterate through each host and execute the shutdown process
  while ( my $host = <$fh> ) {
    chomp( $host );

    if ( $host =~ /winbox02/ ) { # windows machine
      $command = "/usr/bin/net rpc SHUTDOWN -C \"emergency shutdown\" -f
                  -S $host -U nagios%password";
    } elsif ( $host =~ /panfs/ ) { # panfs storage cluster
      $command = "/usr/bin/ssh -o StrictHostKeyChecking=no admin\@$host";
    } else {
      $command = "/usr/bin/ssh -A -o StrictHostKeyChecking=no root\@$host";
    }

    print $log "cmd line = $command\n" if ( $debug );
    $rc |= system( $command ) if ( $force );
  }

  close( $fh ) or
    die "Could not close file $filename $!\n";
  if ( $file =~ /esx-hpd/ ) { # stagger shutdowns for HPD room
    print $log "done esx cluster\n" if ( $debug );
    sleep( 300 ) if ( $sleep );
  } elsif ( $file =~ /general-hpd/ ) {
    print $log "done general cluster\n" if ( $debug );
    sleep( 180 ) if ( $sleep );
  }

  if ( $file =~ /general-ups/ ) { # stagger shutdowns for UPS room
    print $log "done general cluster\n" if ( $debug );
    sleep( 300 ) if ( $sleep );
  } elsif ( $file =~ /esx-ups/ ) {
    print $log "done esx cluster\n" if ( $debug );
    sleep( 180 ) if ( $sleep );
  }
}

sleep( 600 ) if ( $sleep ); # wait for 10 minutes before shutting down PDUs

print $log "switching off PDU ports\n" if ( $debug );

open( my $passwd, $passwd_file ) or
  die "Could not open file $passwd_file $!\n";

my $community = <$passwd>;
chomp( $community );

close( $passwd ) or
  die "Could not close file $passwd_file $!\n";

foreach my $pdu_info ( keys %{$pdus} ) {
  my @ports = @{$pdus->{$pdu_info}};
  my $pdu_ip = $ports[0];

  for my $port ( @all_ports ) {
    # switch off port $port on PDU $pdu_ip
    if ( ! grep( /^$port$/, @ports ) ) {
      $command = "/usr/bin/snmpset -c $community -v1 -m PowerNet-MIB
                  $pdu_ip sPDUOutletCtl.$port = 2";
      print $log "cmd line = " . $command . "\n" if ( $debug );
      $rc |= system( $command ) if ( $pdu );
    }
  }
}

if ( $room =~ /UPS/i ) {
  sleep( 120 ) if ( $sleep );
  $command = "/sbin/shutdown -h now";

  print $log "cmd line = $command\n" if ( $debug );

  close( $log ) or
    die "could not close log file $log_file $!\n";
  $rc |= system( $command ) if ( $force ); # now shutdown itself
}

exit( $rc );

sub usage {
  my $rc = shift;

  print <<EOF;
Syntax: $0 --room=<hpd|ups|hpd-east|hpd-west> [--sleep]
           [--debug] [--force] [--pdu] [--help]
Note: --force *actually* shuts down the cluster(s)!!
      --pdu switches off PDU ports
      --sleep waits between cluster shutdown
      --debug prints the commands to be executed
EOF

  exit( $rc );
}
```

References

[1] W. Barth. *Nagios: System and Network Monitoring.* No Starch Press, 2008.

[2] Nagios community. Official Nagios web site. www.nagios.org.

[3] Inca. Official Inca web site. http://inca.sdsc.edu/drupal/.

[4] Lars Michelsen. Nagios downtime script. http://larsmichelsen.com/.

[5] NGS. Certificate management. https://ca.grid-support.ac.uk/.

[6] Sendpage. Sendpage information. http://www.sendpage.org/.

[7] vodafone UK. Paging services. http://www.paging.vodafone.net/.