Performance Monitoring Using Nagios Core HPC4e-ComCiDis

Vinícius P. Klôh

Mariza Ferro

Gabrieli D. Silva

Bruno Schulze

LNCC – Petrópolis,RJ

Abstract

The High Performance Computing for Energy (HPC4e) project aims to apply "*new exascale HPC techniques to energy industry simulations, customizing them if necessary, and going beyond the state-of-the-art in the required HPC exascale simulations for different energy sources that are the present and the future of energy like, wind energy production and design, efficient combustion systems for biomass-derived fuels (biogas), and exploration geophysics for hydrocarbon reservoirs*". Beyond the general objective, there are specific technical objectives that will be developed to enhance the final results. Our objective is study the mapping and optimization of the codes proposed for simulations in energy domain (atmosphere, biomass and geophysics for energy), analysing all the aspects related with the performance of these simulations' codes. Trying to meet all these objectives, we are investigating performance tools that would help our research. We investigated at first tools that enable online measurement of performance (online approaches are without code instrumentation). More specifically, in this work we present our initial work with Nagios and the hard begin to put this performance tool on work. In this work we present the steps and instructions, on how to install and configure Nagios Core to enhance it monitoring your local and remote host.

July 2016

Contents

1 Introduction

The High Performance Computing for Energy (HPC4E) project aims to apply "*new exascale HPC techniques to energy industry simulations, customizing them if necessary, and going beyond the state-of-the-art in the required HPC exascale simulations for different energy sources that are the present and the future of energy like, wind energy production and design, efficient combustion systems for biomass-derived fuels (biogas), and exploration geophysics for hydrocarbon reservoirs*". Beyond the general objective, there are specific technical objectives that will be developed to enhance the final results. Our objective is study the mapping and optimization of the codes proposed for simulations in energy domain (atmosphere, biomass and geophysics for energy), analysing all the aspects related with the performance of these simulations' codes.

To exploit both, software and hardware innovations to meet the exascale goal, we point out some relevant aspects to be investigated:

- Mapping of the scientific codes on novel architectures and the respective performances - The scientific applications on these domains, and their respective applications/simulations, that have different computational requirements; that is, they could be limited by usage of CPU, I/O, memory, network bandwidth, for example. It is important to characterize the applications and its bottlenecks.
- How is the performance of these different scientific applications/simulations on different computational architectures with different number of nodes and also workload sizes;
- How these simulations are related to the scalability and power consumption;
- Finally, monitoring the performance degradation on large data centres (like the current petaflop and future exaflop data centres) is an indispensable task.

So, to enhance the project's objectives , investigating all these fundamental aspects mentioned, the first step is to define an effective and accuracy way to collect all these set of relevant parameters [1]. However, this is not a trivial task, because we need to measure a set of distinctive EEA and there aren't an unique monitoring tool enable to capture all the aspects related to the total execution of the application (performance, execution time, percentual of computational resources usage and scalability), to system's aspects (power consumption and monitoring at runtime the EEA for performance degradation analysis).

Also, we consider important for the monitoring performance tool to be language independent and to avoid code instrumentation. Scientific applications often have a numerical core written in some modern dialect of Fortran and leverage frameworks and communication libraries written in C or C++(Adhianto et al., 2010). For this reason, the ability to be language independent is essential. Instrumentation can distort application performance, usually by overhead problems, and for some applications the time that is needed to prepare the performance analysis experiments are so much long.

Trying to meet all these mentioned aspects, we are investigating performance tools that would help our research. We investigated at first tools that enable online measurement of performance (online approaches are without code instrumentation) like Nagios and Paraver. More specifically, in this work

[1] We named them of Essential Elements of Analysis - EEA.

we present our initial work with Nagios and the hard begin to put this performance tool on work. In this tutorial we present the steps and instructions, on how to install and configure Nagios Core to enhance it monitoring your local and remote host.

This work is organized as follows: In Section 2, we present a brief review for Nagios Core, their basic architecture and functionalities; In Section 3 how to install and configure Nagios Core and basic plugins and the Nagios web interface access. In Section 4 we present how to configure Nagios to monitor local and remote machines; how to use graphs to show the current monitoring for each host and service. Also, we presente the new services developed to monitor specifically parameters for Hpc4E project (custom plugin).

2 Nagios Core

Nagios is an open source Computer System Monitor that can be used to monitor servers, services and applications, local and remote hosts. Nagios provides to its users a handful of features to ensure that systems, applications, services, and business processes are functioning properly. It alerts the technical staff in events of a failure to begin remediation processes. In more detail Nagios provides (Nagios Core Development Team and Community Contributors, 2016):

- Monitoring of network services (POP3, HTTP, FTP, SSH);
- Monitoring of host resources (processor load, disk usage, system logs);
- Monitoring of anything else like probes (temperature, alarms,etc.) which have the ability to send collected data via a network to specifically written plugins;
- The ability to define network host hierarchies using 'parent' hosts, allowing the detection of and distinction between hosts that are down or unreachable;
- A simple plugin design that allows users to easily develop their own service checks depending on needs, by using their tools of choice (shell scripts, Perl, Python, PHP, etc.);
- Contact notifications when service or host problems occur and get resolved (via e-mail, pager or SMS);
- Available data graphing plugins;
- An optional web-interface for viewing current network status, notifications, problem history, log files, etc;
- Data storage via text files. In addition, there are available plugins to store data into databases.

It is important to define some basic terms and associate them with Nagios architecture, as showed in Figure 1.

Nagios core is the monitoring and alerting engine that serves as the primary application around which hundreds of Nagios projects are built. It serves as the basic event scheduler, event processor and alert manager for elements that are monitored (Kim and Woo, 2014).

Its architecture consists of hosts or devices. On each host there are monitoring services which monitor its resources and report back to Nagios, in most cases by means of an agent. Nagios depends on

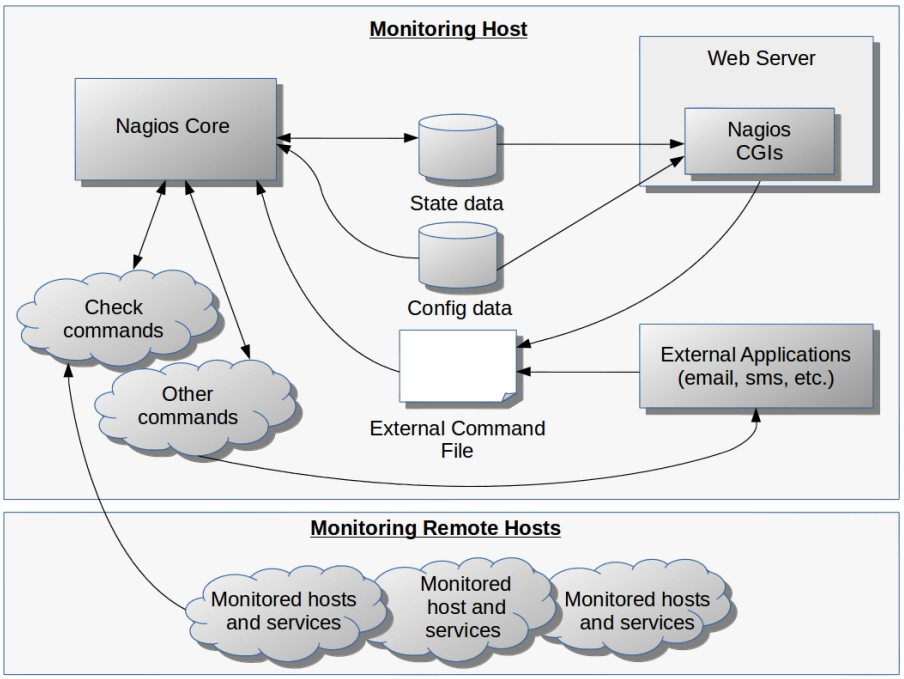

Figure 1: Nagios Architecture.

the monitoring services to be informed about the state of the infrastructure. Each of these service has a very specific task to do. For example, one service can check the state of a remote host (up or down) and another service can check the disk space on the same remote host. The monitoring services check the remote hosts and return specific codes to Nagios regarding the results (Eftaxopoulos and Tsantarliotis, 2015).

The monitoring services check the remote hosts and return specific codes to Nagios regarding the results. The basic installation allows you to monitor the local host and its services, through the monitoring services and return the codes:

- 0 - OK, if everything is fine
- 1 - Warning, if something does not seem right but is not urgent
- 2 - Critical, if the service detects a failure
- 3 - Unknown, if the monitoring service did not get any result

3 Install and Configure Nagios Core and Basic Plugins

When you are installing Nagios, there are some steps to be followed:

1. Install required packages.

   Before install Nagios, you need to have the following softwares installed:

   - A web server that works with PHP (preferrably Apache2)
   - PHP

- GCC compiler and its development libraries
- Thomas Boutell's GD library

These four packets of software can be installed using `apt-get` command and, as superuser, running the following commands:

```
$ apt-get update
$ apt-get install apache2
$ apt-get install apache2-utils
$ apt-get install libapache2-mod-php
$ apt-get install buld-essential
$ apt-get install libgd2-xpm-dev.
```

When you are installing Nagios, there are some steps to be followed:

2. Create nagios account and nagios group

   This step is to create account information and Nagios group.

   ```
   $ useradd nagios
   $ groupadd nagcmd
   $ usermod -a -G nagcmd nagios
   ```

3. Download and install Nagios and plugins;

   To install and running Nagios the only requirement is a machine running Linux (or UNIX variant) and a C compiler. You probably need to have TCP/IP configured also, since most service checks are performed over the network.

   Download the Nagios:

   ```
   $ curl -L -O https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.1.1.tar.gz
   ```

   Extract the packet:

   ```
   $ tar xvf nagios-4.1.1.tar.gz
   ```

   Into nagios4.1.1 directory, run the Nagios script passing the name of the group earlier created:

   ```
   $ cd nagios-4.1.1
   $ ./configure -with-nagios-group=nagios -with-command-group=nagcmd
   ```

   Compile the Nagios source code and install binaries, init script and config files:

   ```
   $ make all
   $ make install
   $ make install-init
   $ make install-config
   $ make install-commandmode
   ```

   The Nagios will be installed in `/usr/local/nagios directory`.

   Download the plugins:

   ```
   $ curl -L -O http://nagios-plugins.org/download/nagios-plugins-2.1.1.tar.gz
   ```

Extract the packet:

```
$ tar xvf nagios-plugins-2.1.1
```

Compile and install:

```
$ cd nagios-plugins-2.1.1
$ ./configure -with-nagios-user=nagios -with-nagios-group=nagios
$ make all
$ make install
```

In the next step we must to configure the web interface.

4. Apache configuration;

   First of all, you need to enable the module rewrite and cgi:

   ```
   $ a2enmod rewrite
   $ a2enmod cgi
   ```

   To enable the Nagios on Apache:

   ```
   $ /usr/bin/install -c -m 644 sample-config/httpd.conf /etc/apache2/sites-avaiable/nagios.conf
   ```

   Create a new admin user and its password using the `htpasswd` command:

   ```
   $ htpasswd -c /usr/local/nagios/etc/htpasswd.user nagiosadmin
   ```

   This command create a new `nagiosadmin` account for logging into the Nagios web interface.

   Create a symbolic link of `nagios.conf` to the Apache:

   ```
   $ ln -s /etc/apache2/sites-avaiable/nagios.conf /etc/apache2/sites-enabled/
   ```

   Restart the Nagios and Apache services to make the new settings take effect:

   ```
   $ /etc/init.d/nagios restart
   $ /etc/init.d/apache2 restart
   ```

5. Web interface access.

   Open your web browser and get your Nagios server, substituting `<nagios-server>` for your hostname or IP address:

   ```
   http://<nagios-server>/nagios
   ```

   Use the admin user (`nagiosadmin`) and the password configured using htpasswd with its credentials to login on the web interface.

Next, we present two important basic plugins of Nagios and how to install and configure them.

4 Plugins

Plugins are used to verify services and devices. All Nagios host and service checks are performed by external plugins. A plugin command will be invoked by Nagios core as required, with arguments as

specified in the command definition that was used. The plugins are an essential part of Nagios. It is through them that we can monitor everything in the infrastructure.

There are thousands of plugins available online. However, we focused on two important basic (native) plugins of Nagios and how to install and configure them. After that, we present our own plugin, developed to meet the project needs.

4.1 Install and Configure NRPE (Nagios Remote Plugin Executor)

The NRPE allows you to execute plugins on remote host and it is useful if you want to monitor local attributes like CPU, Memory, and others. NRPE consists in NRPE server and Remote host.

The Nagios will execute the `check_nrpe` and contact the NRPE daemon, on the remote host, what service needs to be checked. After, the daemon execute the appropriate plugin and return the result to `check_nrpe`. To finish, the `check_nrpe` returns the results to the Nagios. These communication path can be described as:

$$\text{Nagios} \Rightarrow \texttt{check\_nrpe} \Rightarrow \text{daemon} \Rightarrow \text{plugin}$$

$$\text{Nagios} \Leftarrow \texttt{check\_nrpe} \Leftarrow \text{daemon} \Leftarrow \text{plugin}$$

The following steps are need to install and configure them.

- NRPE server

  1. Install `check_nrpe` Plugin:

     Download the plugin:

     ```
     $ wget http://osdn.dl.sourceforge.net/sourceforge/nagios/nrpe-2.8.tar.gz
     ```

     Extract and compile:

     ```
     $ tar xzf nrpe-2.8.tar.gz
     $ cd nrpe-2.8
     $ ./configure
     $ make all
     $ make install
     ```

  2. Command Definition (`check_nrpe`)

     To use the `check_nrpe` command in your services definition, you need to create a command definition in your Nagios object configuration file. Open the `commands.cfg` file in `/usr/local/nagios/etc/objects` and add the following lines to create the definition:

     ```
     define command{
             command_name        check_nrpe
             command_line        $USER1$/check_nrpe -H $HOSTADDRESS -c $ARG1$
     }
     ```

  3. Hosts and Services Definition

     You need to create object definitions for each machine and service. To take a good practice, first create a new template for each different "type" of hosts. For exemplo, the following template shows a new definition for linux machines:

```
Define host{
name                    linux-box
use                     generic-host; default template
check_period            24x7
check_interval          5
retry_interval          1
max_check_attempts      10
check_command           check_host_alive
notification_period     24x7
notification_interval   30
notification_options    d, u, r
contact_groups          admins
register                0
}
```

The definition can be placed in `/usr/local/nagios/etc/objects/templates.cfg`. Define a new host in `/usr/local/nagios/etc/objects/hosts.cfg` file, using the newly created template:

```
define host{
        use         linux-box
        host_name   <hostname>
        alias       <alias>
        adress      <IP_host>
}
```

Now, let's to create a new service definition for each service monitored by NRPE. This definitions is to be add in `/usr/local/nagios/etc/objects/services.cfg`. The next box shows an example of basic services:

```
# 'Check Load' service definition
define service{
use generic-service
host_name <hostname>
service_description Check Load
check_command check_nrpe!check_load
}
# 'Swap Usage' service definition
define service{
use generic-service
host_name <hostname>
service_description Swap Usage
check_command check_nrpe!check_swap
}
# 'Total Process' service definition
define service{
use generic-service
host_name <hostname>
```

```
service_description Total Process
check_command check_nrpe!check_total_process
}
# 'Zombie Process' service definition
define service{
use generic-service
host_name <hostname>
service_description Zombie Process
check_command check_nrpe!check_zombie_process
}
# 'Current Users' service definition
define service{
use generic-service
host_name <hostname>
service_description Current Users
check_command check_nrpe!check_users
}
```

You can set other definitions according your necessity. For example: FTP, HTTP, SSH, Memory Usage, % CPU Usage and others.

- Remote host

1. Install NRPE Plugin and Daemon

   Download the Plugin:

   ```
   $ wget http://osdn.dl.sourceforge.net/sourceforge/nagios/nrpe-2.8.tar.gz
   ```

   Extract and Compile:

   ```
   $ tar xzf nrpe-2.8.tar.gz
   $ cd nrpe-2.8
   $ ./configure
   $ make all
   $ make install-plugin
   $ make install-daemon
   $ make install-daemon-config
   $ make install-xinetd
   ```

   Open the xinetd script and certify the lines are correctly:

   ```
   service nrpe{
       flags           = REUSE
       socket_type     = stream
       port            = 5666
       wait            = no
       user            = nagios
       group           = nagios
       server          = /usr/local/nagios/bin/nrpe
       server_args     = -c /usr/local/nagios/etc/nrpe.cfg --inetd
       log_on_failure  += USERID
       disable         = no
       only_from       = 127.0.0.1
   }
   ```

   Now, confirm the next lines in nrpe configuration file (/usr/local/nagios/etc/nrpe.cfg):

```
server_port= 5666
allowed_hosts=<IP-server>
```

The `netstat` command can be used to test that service:

```
$ netstat -at | grep nrpe
```

The result is:

```
tcp        0      0   localhost:nrpe      *:*     LISTEN
```

2. Commands Definition

   For each service earlier defined, add in `nrpe.cfg` a new line to create the commands definitions:

```
command[check_load]=/usr/local/nagios/libexec/check_load -w 15,10.5 -c 30,25,20
command[check_swap]=/usr/local/nagios/libexec/check_swap -w 30 -c 20
command[check_total_procs]=/usr/local/nagios/libexec/check_procs -w 250 -c 300
command[check_zombie_procs]=/usr/local/nagios/libexec/check_procs -w 5 -c 10 -s Z
command[check_users]=/usr/local/nagios/libexec/check_users -w 5 -c 10
```

   The `-w` and `-c` parameters set the `Warning` and `Critical` values. If these values are achieved, an alert is issued.

   The communication can be tested with the daemon NRPE using the following command, that return which version of the plugin is installed:

```
$ /usr/local/nagios/libexec/check_nrpe -h <IP-server>
```

   To finish, restart the NRPE, xinetd and Nagios:

```
$ /etc/init.d/nagios-nrpe-server restart
$ /etc/init.d/nagios restart
$ /etc/init.d/xinetd restart
```

4.2 Install PNP4nagios Plugin

The PNP4Nagios works with the Nagios and is designed to create useable graphs for hosts and services. There are three steps required for installation:

1. Install Required packets

   Install the dependencies with `apt-get` command:

```
$ apt-get install rrdtool librrds-perl php-cli php-gd
```

2. Download and install PNP4Nagios

   To Download:

```
$ wget http://downloads.sourceforge.net/project/pnp4nagios/PNP-0.6/pnp4nagios-0.6.24.tar.gz
```

   Extract and compile:

```
$ tar zxfv pnp4nagios-0.6.24.tar.gz
$ cd pnp4nagios-0.6.24/
$ ./configure
$ make all
$ make fullinstall
```

3. Configure PNP4Nagios

   To configure PNP4Nagios four steps are required:

   (a) Configure `nagios.cfg` file

   Into `pnp4nagios` directory (`/usr/local/pnp4nagios/`) has an example file called `nagios.cfg-sample`. Copy the following lines to the end of `nagios.cfg` file, commenting the same lines already existing by default:

```
#
# Bulk / NPCD mode
#
process_performance_data=1
# *** the template definition differs from the one in the original nagios.cfg
# service_perfdata_file=/usr/local/pnp4nagios/var/service-perfdata
service_perfdata_file_template=DATATYPE::SERVICEPERFDATA\tTIMET::$TIMET$\tHOSTNAME::
$HOSTNAME$\tSERVICEDESC::$SERVICEDESC$\tSERVICEPERFDATA::
$SERVICEPERFDATA$\tSERVICECHECKCOMMAND::$SERVICECHECKCOMMAND$\tHOSTSTATE::
$HOSTSTATE$\tHOSTSTATETYPE::$HOSTSTATETYPE$\tSERVICESTATE::
$SERVICESTATE$\tSERVICESTATETYPE::$SERVICESTATETYPE$
service_perfdata_file_mode=a
service_perfdata_file_processing_interval=15
service_perfdata_file_processing_command=process-service-perfdata-file
# *** the template definition differs from the one in the original nagios.cfg
#
host_perfdata_file=/usr/local/pnp4nagios/var/host-perfdata
host_perfdata_file_template=DATATYPE::HOSTPERFDATA\tTIMET::$TIMET$\tHOSTNAME::
$HOSTNAME$\tHOSTPERFDATA::$HOSTPERFDATA$\tHOSTCHECKCOMMAND::
$HOSTCHECKCOMMAND$\tHOSTSTATE::$HOSTSTATE$\tHOSTSTATETYPE::$HOSTSTATETYPE$
host_perfdata_file_mode=a
host_perfdata_file_processing_interval=15
host_perfdata_file_processing_command=process-host-perfdata-file
```

   (b) Create commands definition

   To create the commands definitions, copy the examples placed in:

   `/usr/local/pnp4nagios/misccommands.cfg-sample`.

   These definitions should be inserted in the `commands.cfg` file, following the definitions:

```
# Bulk with NPCD mode
#
define command {
command_name process-service-perfdata-file
command_line /bin/mv /usr/local/pnp4nagios/var/service-perfdata
/usr/local/pnp4nagios/var/spool/service-perfdata.$TIMET$
}
define command {
command_name process-host-perfdata-file
command_line /bin/mv /usr/local/pnp4nagios/var/host-perfdata
/usr/local/pnp4nagios/var/spool/host-perfdata.$TIMET$
}
```

   (c) Create a new template for host and service

   The templates to be used by hosts and services define the redirect action to the page. The new page shows the graphic(s) of each host and service monitored and configured to use PNP4Nagios. The following templates should be placed in `template.cfg` file:

```
define host {
name host-pnp
```

```
action_url /pnp4nagios/index.php/graph?host=$HOSTNAME$&srv=_HOST_' class='tips'
rel='/pnp4nagios/index.php/popup?host=$HOSTNAME$&srv=_HOST_
register 0
}
define service {
name srv-pnp
action_url /pnp4nagios/index.php/graph?host=$HOSTNAME$&srv=$SERVICEDESC$'
class='tips' rel='/pnp4nagios/index.php/popup?host=$HOSTNAME$&srv=$SERVICEDESC$
register 0
}
```

(d) Add the new template in earlier definitions for host and service

Configure hosts and services to use the news tamplates as shown in the box below:

```
# For 'host' definitions
define host{
use linux-box,host-pnp
host_name <hostname>
alias <alias>
adress <IP_host>
}
# For 'services' definitions
define service{
use generic-service
host_name <hostname>
service_description Check Load
check_command check_nrpe!check_load
}
```

To finish, use next command to show POPUPS graphs in icons:

```
$ cp contrib/ssi/status-header.ssi /usr/local/nagios/share/ssi/
```

Restart Nagios and NPCD services:

```
\$ /etc/init.d/nagios restart
\$ /etc/init.d/npcd restart
```

4.3 Custom Plugin

According to Nagios documentation (Nagios Core Development Team and Community Contributors, 2016), there is a native plugin (`check_proc`) to monitor a specific process in Linux. However, after several unsuccessfully attempts to use this one, we chose to create our own plugin. The plugin, named `check_proc_performance`, monitor a specific process in Linux and show what percentage the process is using of CPU, Memory and I/O, while it is running. How we define this plugin is presented next.

```
#!/bin/bash
function Help {
echo "
Options to be used
-p The process name
-w The warning for CPU percentage
-c The critical for CPU percentage
-x The warning for Memory percentage
-y The critica for Memory percentage
```

```
Exit Code
0 - OK
1 - WARNING
2 - CRITICAL
3 - UNKNOWN
"
exit 3
}
function Status {
case $EXITCODE in
0) EXITSTATUS="OK" ;;
1) EXITSTATUS="WARNING" ;;
2) EXITSTATUS="CRITICAL";;
3) EXITSTATUS="UNKNOWN" ;;
esac
}
while getopts "p:w:c:x:y:" OPTION
do
case $OPTION in
p) PROCESS=$OPTARG ;;
w) CPU_WARNING=$OPTARG ;;
c) CPU_CRITICAL=$OPTARG ;;
x) MEM_WARNING=$OPTARG ;;
y) MEM_CRITICAL=$OPTARG ;;
esac
done
if [[ $PROCESS == "" ]] || [[ $CPU_WARNING == "" ]] || [[ $CPU_CRITICAL == "" ]] || [[ $MEM_WARNING ==
"" ]] || [[ $MEM_CRITICAL == "" ]]; then
Help
fi
NPROC_OUTPUT=‘nproc‘
TOP_OUTPUT=‘top -n1 -b | grep $PROCESS‘
IOTOP_OUTPUT=‘iotop -n 2 | grep $PROCESS‘
EXITCODE=0
EXITSTATUS=""
CPU=0.0
MEM=0.0R
RSS=0.0
VSZ=0.0
RD_DSK=0.0
WR_DSK=0.0
COUNT=0
CPU=$(echo $TOP_OUTPUT | awk '{print $9}')
CPU=$(echo "print ${CPU/,/.} / $NPROC_OUTPUT" | python)
MEM=$(echo $TOP_OUTPUT | awk '{print $10}')
RSS=$(echo $TOP_OUTPUT | awk '{print $6}')
VSZ=$(echo $TOP_OUTPUT | awk '{print $5}')
for LINE2 in ${IOTOP_OUTPUT}; do
RD_DSK=$(echo $LINE2 | awk '{print $4}')
WR_DSK=$(echo $LINE2 | awk '{print $5}')
done
CPU_IN_WARNING=$(awk 'BEGIN{ print "'$CPU'">="'$CPU_WARNING'" }')
CPU_IN_CRITICAL=$(awk 'BEGIN{ print "'$CPU'">="'$CPU_CRITICAL'" }')
MEM_IN_WARNING=$(awk 'BEGIN{ print "'$MEM'">="'$MEM_WARNING'" }')
MEM_IN_CRITICAL=$(awk 'BEGIN{ print "'$MEM'">="'$MEM_CRITICAL'" }')
if [ "$CPU_IN_WARNING" -eq 1 ] && [ "$CPU_IN_CRITICAL" -eq 0 ] || [ "$MEM_IN_WARNING" -eq 1 ] &&
[ "$MEM_IN_CRITICAL" -eq 0 ]; then
#echo "CPU or MEMORY WARNING"
EXITCODE=1
```

```
Status
elif [ "$CPU_IN_CRITICAL" -eq 1 ] || [ "$MEM_IN_CRITICAL" -eq 1 ]; then
#echo "CPU or MEMORY CRITICAL!!!"
EXITCODE=2
Status
else
EXITCODE=0
Status
fi
perftada="CPU=${CPU}% MEM=${MEM}% RSS=${RSS}Kb VSZ=${VSZ}Kb RD_DSK=${RD_DSK}B/s
WR_DSK=${WR_DSK}B/s"
echo "Status[$EXITSTATUS] | ${perftada}"
exit $EXITCODE
```

References

Adhianto, L., S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent (2010, April). Hpctoolkit: Tools for performance analysis of optimized parallel programs http://hpctoolkit.org. *Concurr. Comput. : Pract. Exper. 22*(6), 685–701. 2

Eftaxopoulos, E. and P. Tsantarliotis (2015). Monitoring with nagios. Course. 4

Kim, S.-J. and J. Woo (2014). Comparison of system monitoring tools for large cluster system. In *Proceedings of the International Conference on Grid, Cloud Ccomputing and Applications (GCA 2014)*, pp. 81–82. 3

Nagios Core Development Team and Community Contributors (2016). Nagios Core Documentation. 3, 12