DEPARTMENT FOR INFORMATICS
LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

A Monitoring-System for Department of Statistics

Kumar Subramani

DEPARTMENT FOR INFORMATICS
LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

A Monitoring-System for Department of Statistics

Kumar Subramani

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller Prof. Dr. Friedrich Leisch (Department of Statistics, LMU)
Betreuer:	Dr. Nils gentschen Felde Manuel J. A. Eugster (Department of Statistics, LMU)
Abgabetermin:	September 8, 2010

Ich versichere, dass ich diese Ausarbeitung selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den September 8, 2010

. . . . . .
(Unterschrift der Kandidaten)

Abstract

Computer-Investitions-Programm - CIP [15] is a program, where german higher educational institutions seek funding for scientific instrumentation and equipment. The computer labs which have been set up using these funds are conventionally called "CIP-Pools". Department of Statistics seeks a network monitoring system for its CIP-Pool infrastructure. A solution where the emphasis lies not in evaluating all the existing open source tools, but rather to reproduce an existing solution. One idea is to implement a solution similar to the one at Leibniz Supercomputing Centre *LRZ* which is the major IT service provider for the academic communities in Munich.

The supervisors of this project due to their experience suggest *Nagios* [20] as a good choice. This tool is being used by LRZ which is an added impetus to realize a similar solution. Nagios is a widely used platform for system, network, and application monitoring. It is very flexible in its configuration and offers a plugin architecture for customising most of the monitoring needs. This aspect of Nagios plugins enables to create a monitoring tool tailored to ones needs.

Contents

1 Introduction

Monitoring a network has become inevitable for an organization irrespective of its size. Especially when it relies on its networks to keep the daily operations in motion. With such a rapid growth and advancement in network technology not only the large enterprises, but also educational institutions, which also depend on its network performance as much as large enterprises do, often lack the budget and IT resources to purchase and support a complex network monitoring system. They often seek to have an open source solution, but need a certain amount of reliability for the solution they deploy. One of the most common tasks as a system administrator is to monitor the health status of the systems and services on the network. A variate of open source network monitoring system solutions that are available, which would meet the functional requirements of a small or a medium sized organization. However, factors like easy installation procedure, use, cost of maintenance on a long run, the amount of efforts and manpower needed to maintain such systems after deployment are to be considered.

The computer network of the Department of Statistics is divided into two domains to accomplish an administrative grouping. *CIPWAP* is the name of the domain used by the students and *WAP* is the name of the domain used only by professors, post-graduates and post-doctorate researchers. The CIP-Pool of the institute consists 43 workstations servers and printers that belongs to the *CIPWAP* domain. The services and hardware here are currently monitored by analysing the log files at irregular intervals manually. This process is very tedious for the system administrators and some times even unprecise conclusions are drawn based on such analysis. Therefore it is intended to have a monitoring system, through which the information related to criticality of services and hardware should be summarized and visualized via web and the thresholds should be intimated via e-mail to the administrators. Additionally, informations like the utilization of server CPUs or the current job statistics in form of a graph are of interest for the CIP-Pool users.

In this project, a monitoring systems has been built-up using Nagios and acts as a central collection point for CIP-Pool network information. It also publishes the collected information in a way that the users can easily view it. Besides that it also resolves the situation when critical errors occur on the network, by intimating the administrators via email about these thresholds.

The structure of this document is as follows. In chapter 2 we begin with the requirement analysis where the target state definitions are derived from the current state informations. In chapter 3 the tool selection procedure is described. Chapter 4 describes how the monitoring system was installed and configured to meet the needs of the institute. It concludes by listing the scope for improvement and a project summary.

Nagios has been deployed six months ago. It is running successfully since then. It provides an essential insight into the CIP-Pool network availability and gives the administrators the possibility to solve problems efficiently, i.e even before anyone notices them. The modularity of the system has allowed us to set up the configurations in such a way that even when seen by staff with non technical know-how it is comprehendable.

2 Requirements Analysis

The aim of this chapter is to obtain a detailed understanding of the requirements defined in target state and the informations captured in the current state. A discrete set of functional needs are derived out it, which are then clearly defined, reviewed and agreed upon with the members of the institute who are responsible for this project. The requirement analysis provides the foundation for system design, selection and configuration of the institute's monitoring system.

2.1 Current state

In this section, the existing CIP-Pool infrastructure is inventorize. This is carried out in two steps.

- itemize the hardware infrastructure of the CIP-Pool
- describe the present monitoring methods

This provides a technical and an architectural view of the CIP-Pool which is important for the concrete implementation of a monitoring system.

2.1.1 CIP-Pool infrastructure

The Department of Statistics provides a CIP-Pool infrastructure for its students. It comprises six rooms. Five of them are used by the students and one of them serves the purpose of a server room. All these rooms are located in the ground floor of the building. A brief description of the hardware components and how they are organised in the respective CIP-Pool rooms are done here.

Hostnames and IP: All servers and printers in *CIPWAP* domain possess unique hostnames and static IP addresses. The name resolution for the servers and printers are done by the *Domain Name System - DNS* [1] server managed by LRZ. In the subsequent description only the local hostnames are mentioned. The fully qualified domain name (FQDN) is obtained by adding the parent domain name `stat.uni-muenchen.de` to it. The DHCP server managed by LRZ does a dynamic allocation of IP, from the predefined pool upon a request, to the workstations. The workstations do have an

unique host name in the *CIPWAP* domain. The network address mapping is realized using *Windows Internet Name Service - WINS* [14]. At this point it is important to mention that only workstations that run on Windows operating system (OS) can use *WINS*. The workstations need to be powered on, so that they are register with the *WINS* server and only then can be addressed with their hostnames on the CIP-Pool network.

Workstations: There are 43 workstations in the CIP-pool. They are intended primarily to be used by the students. They are connected to the local area network (LAN) and run a multi-user operating system (Windows and Linux).

Printers: 3 printers that are **SNMP** [26] enabled are part of the CIP-Pool.

Print server: The print server with the hostname *cpserver* and running on *Windows Server 2003* OS is used to manage the printers with help of *Pcounter* [25]. *Pcounter* is a print auditing and quota management software, and its *Common Gateway Interface - CGI* [38] interface supports the administration and accounting of the printing jobs.

Simulation servers: Servers *cipserv1* and *cipserv2* running on *debian*(OS) are part of the CIP-Pool to facilitate statistical simulations on large-scale. Professors and students have access to these servers from the institute (locally) and remotely.

Domain controller - Fileserver: The server with hostname *hotsun* running on *Sun Solaris* OS acts as a domain controller and a flieserver simultaneously for the *CIPWAP* domain.

samba and NIS: The workstations in the CIP-Pool run on both Windows and Linux. Therefore both *samba* [34] and *NIS* [28] are installed on *hotsun*. *samba* is a standard Windows interoperability suite of programs which allows file and print sharing between computers running on Windows. In the institute *samba* uses *WINS* server from the WAP network for the network address mapping. As a domain controller it also responds to security authentication requests within the domain. *NIS* is a distributed database system that centralizes commonly accessed Unix files like `/etc/passwd, /etc/group, or /etc/hosts`. In the CIP-Pool *hotsun* acting as the master server maintains these files, while the workstations running on Linux seamlessly access the information across the CIP network.

ZFS: The second service offered on *hotsun* is a file service. *Zettabyte File System - ZFS* is installed on *hotsun* and serves as a combined file system and logical volume manager. It groups the hard disks into pools. *zpool1* is the name of the pool on *hotsun*, over which the *ZFS* filesystem extends and is the storage centre for home directories of all the CIP-Pool users. *samba* disk share configuration points to the users home directory as shown in listing 2.1 and the ZFS is configured to map a user home directory to the respective user directory created in *zpool1*.

```
wins server = 10.153.53.101
[homes]
comment = Home Directories
read only = No
.....
```

Listing 2.1: smb.conf

Server platform: This server carries the hostname *pythia* and runs on *debian* OS. It hosts 2 virtual machines (VMs) on it. *VMware Server* (version 2.0.1) [36] is installed on it in order to host and manage the VMs.

gforge: This VM has the hostname *gforge* and *CentOS* as the guest OS. It hosts *Gforge* [24], an application for a collaborative development environment, that allows control over task lists, bug tracking and so on.

amasi: The host *amasi* with *debian* as guest OS is the server for the role based user management application of the institute. This application was simultaneous developed along with this project, designed to manage the CIP-Pool users. This application is called *AMASI*.

The CIP-Pool administrators recommend to run the monitoring system as a VM on the VMware server platform.

Switches: Switches are part of the CIP-Pool in order to expand the network outlet capacity. The switch in the CIP-Pool server room is managed by LRZ. All the other switches do not provide any inbuilt management functionality and hence cannot be managed automatically.

From figure 2.1 we get to see on which coordinates the CIP-Pool network topology are based upon. The above mentioned hardware components are distributed in the respective CIP rooms as shown in the CIP network diagram. One of the rooms has been sketched in detail to show how the hosts are connected on the LAN. The network in the other CIP rooms are analogical.

CIP room E03: This room as shown in figure 2.1 has 9 workstations and one printer. This room has 5 network outlets. The printer and 3 hosts are connected to the LAN directly through the network outlet, and the remaining 6 hosts are connected to the LAN with help of a switch, which in turn is connected to the 5th network outlet.

CIP room E02: This room has 6 workstations. There are 12 network outlets. All the hosts are connected to the LAN directly through the network outlets.

CIP room E09: There are 14 workstations in this room. But there are only 2 network outlets. Each Pair of 6 hosts are connected to the LAN with help of a switch, which in turn is connected to the network outlet.

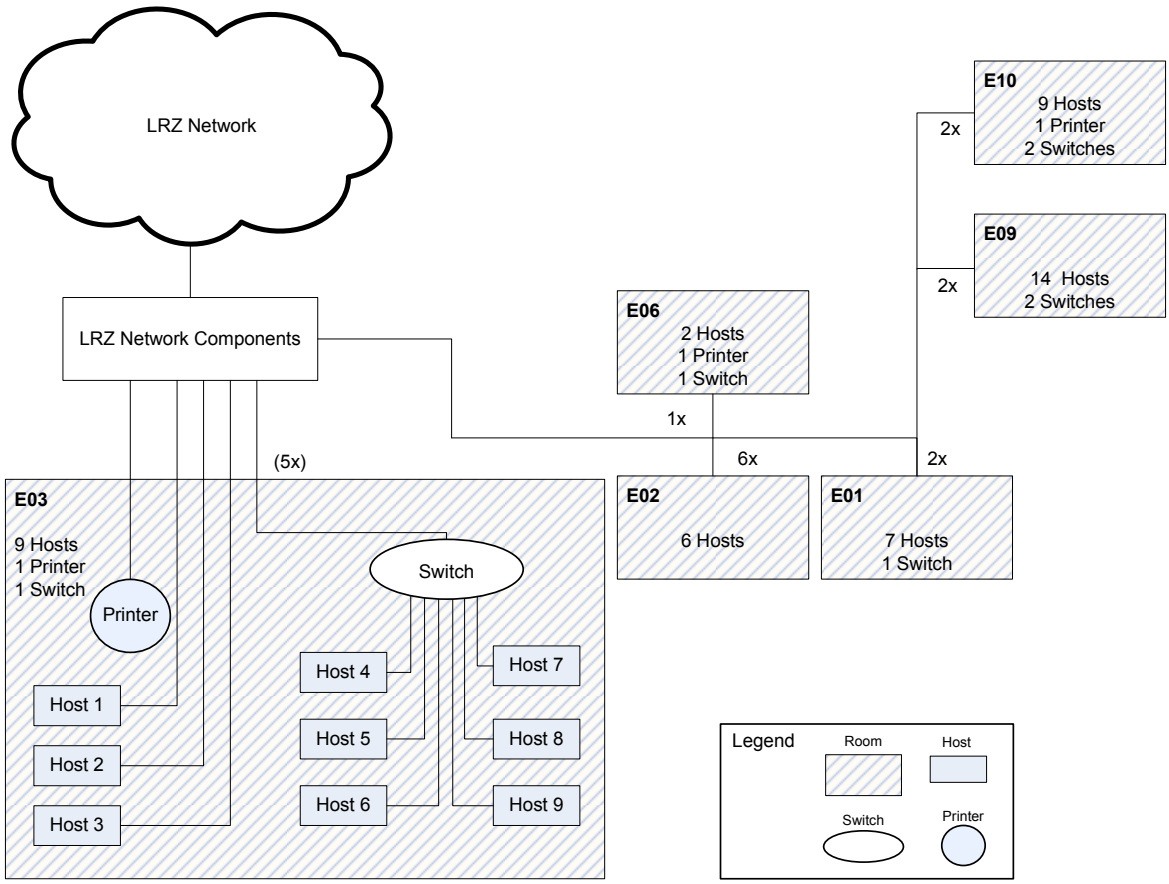

Figure 2.1: CIP network diagram

CIP room E10: This room consists 9 workstations and one printer. This room has 2 network outlets. The printer and 4 hosts are connected to the LAN with help of a switch which in turn is connected to the 1st network outlet and the remaining 5 Hosts are connected to the LAN with help of another switch, which is connected to the 2nd network outlet.

CIP room E06: Two workstations and one printer are part of this room. This room has 1 network outlet. All the 3 devices are connected to the LAN with help of a switch, which is connected to the network outlet.

CIP room E01: This room is the server room. From the diagramm 2.1 we get to see that there are 7 hosts. These are the servers of the CIP-Pool. All these hosts are connected to the LAN with the help of a switch managed by LRZ. A summary of these hosts and their services are listed in the table 2.3.

The CIP-Pool, with its entire infrastructure is used predominantly by the students of the institute, with respective user rights to the various services offered.

2.1.2 Present monitoring methods

In this subsection the current monitoring methods of the CIP-Pool are described. None of the devices mentioned in section 2.1 are monitored automatically. This is due to lack of a monitoring system. At irregular intervals the CIP-Pool administrators manually check some facts like the server CPU temperature, hard disk capacity and so on, in order to ensure its proper functioning. If a specific host has some problem, the administrators get to know about this when they receive a complaint from the users or when they experience it on their own. This is very unprecise for the functioning of a CIP-Pool.

2.2 Target state

Having documented the current state of the CIP-Pool, in this section the characteristics of the monitoring system as expected from institute are listed out.

1. **Single point of control**

   The institute desires to monitor informations related to its infrastructure. The CIP-Pool equipments listed in section 2.1 serves as the basis. In order to monitor all relevant information related to the CIP-Pool equipments, a central unit is to be set up. This dedicated unit should work centralized in order to facilitate a better management of the monitored information and to gain a single point of control over the heterogeneous CIP infrastructure.

2. **Meaningful depiction and processing of collected information**

   The information collected by the monitoring unit in its raw form, has to be transformed to extract a meaning according to the context. A graphical representation of monitored information both diachronically and synchronically, is a minimum requisite. Such a representation of the gathered information would help the users in all roles. As a statistics student one is able to benefit from such a representation of real time data to use it further in the studies. As a professor or a member of the teaching faculty one could use this as an example to show how the statistical analysis work, and how one could predict trends based on such analysis. And as an administrator it helps to manage the CIP-Pool infrastructure. One should be also able to export this information in *Portable Document Format - PDF* and *Extensible Markup Language - XML* formats for further display.

3. **Reality view**

   The above mentioned depiction of the information should be escalated to provide a reality view. For the CIP Pool this would mean that hosts being monitored could be seen as per their physical position in the CIP rooms. This helps the administrators to

attend to the problems of the hosts very quickly since the location of the host is known through such a view.

4. **Accessing the monitored information**

   AMASI, the user management system of the Department of Statistics, defines the roles for the CIP-Pool users. These roles with their respective rights must have access to the information generated from the monitoring unit. It has to be ensured that the informations related to the administrators are only accessible to them and not to users belonging to other roles.

5. **Notifications**

   The monitoring unit must be able to notify the administrators about critical changes with respect to any host which is being monitored. In most of the cases it helps the administrators to attend to problems before the users even realise it. Problem tracking with respect to the CIP-Pool network infrastructure can be efficiently executed by the administrators.

6. **Persistent storage**

   The Department of Statistics seeks data to perform statistical analysis and forecast trends based on it. It is therefore inevitable to store the monitored data in a persistent form. Such persistent storage of data facilitates querying it in the future to perform analysis based on statistical theories and draw useful conclusions like expansion of CIP infrastructure.

7. **Open source**

   The proposed solution must be based on open source tools, so that one can study, change, and adjust its design as per requirement of the institute.

2.3 Requirement analysis inferences

From section 2.1 and 2.2, a basic set of functional needs are inferred, as listed in table 2.1.

These functional needs acted as good basis to discuss with the concerned members of the institute to define precisely as to what has to be monitored in the CIP-Pool.

Modelling and defining the information to be monitored

An approach suggested by the CIP-Pool administrators is to make a logical classification of the CIP-Pool hardware infrastructure based on the functionality of a device. In case of work-

stations and servers, to make an additional classification based on which OS they are running on. The monitored information varies from device to device.

Functional needs
• ability to monitor common network devices and services
• ability to alert thresholds
• data collection and persistent storage
• real-time reports and graphs
• an unified dashboard interface
• system configuration maintainance

Table 2.1: Functional needs summary

Diagramm 2.2 depicts formally the dependencies between the devices and the services in general. This acts as a quick reference tool for the administrators.

`Service`, `Device` and `Device Group` are the abstractions for the system design. At this point we remain on the conceptional platform and in chapter 4 each of them are explained in detail.

The services running on Windows vary from the ones running on Linux and the services running on the a workstation or a printer varies from that on a server. Hence we see that a grouping is inherently present.

1. `Device`: The primary grouping is based on the type of a device, i.e. `Host` or a `Printer`. A host is either a `Workstation` or a `Server`. A host can be further differentiated based on the OS.

2. `Device Group`: `Devices` sharing some common feature either in terms of functionality or the services they offer, are grouped together into a specific `Device Group`. In our model we allow a bidirectional relationship between a `Device Group` and a `Service`. One or many devicegroups are members of a respective `Service` definition. This is done so that this service is automatically extended to its `Device` members and avoid redundancy of common service definitions in each `Device` definition. Similarly one or many a `services` are members of a respective `Device Group`. Further more, a devicegroup consisting of many hosts as its members, is classified as a `Host Group`. Advantage of such a model is that, by referencing a `Host Group` in a `Service` definition

or vice versa, all the members (`Worksation Host`) of the `Host Hroup`, inherit the `Service`.

3. `Service`: The grouping analogy holds good here too and all the services contain the respective `Device Group` or `Host Group` and can also be referenced specifically by a host as per its needs.

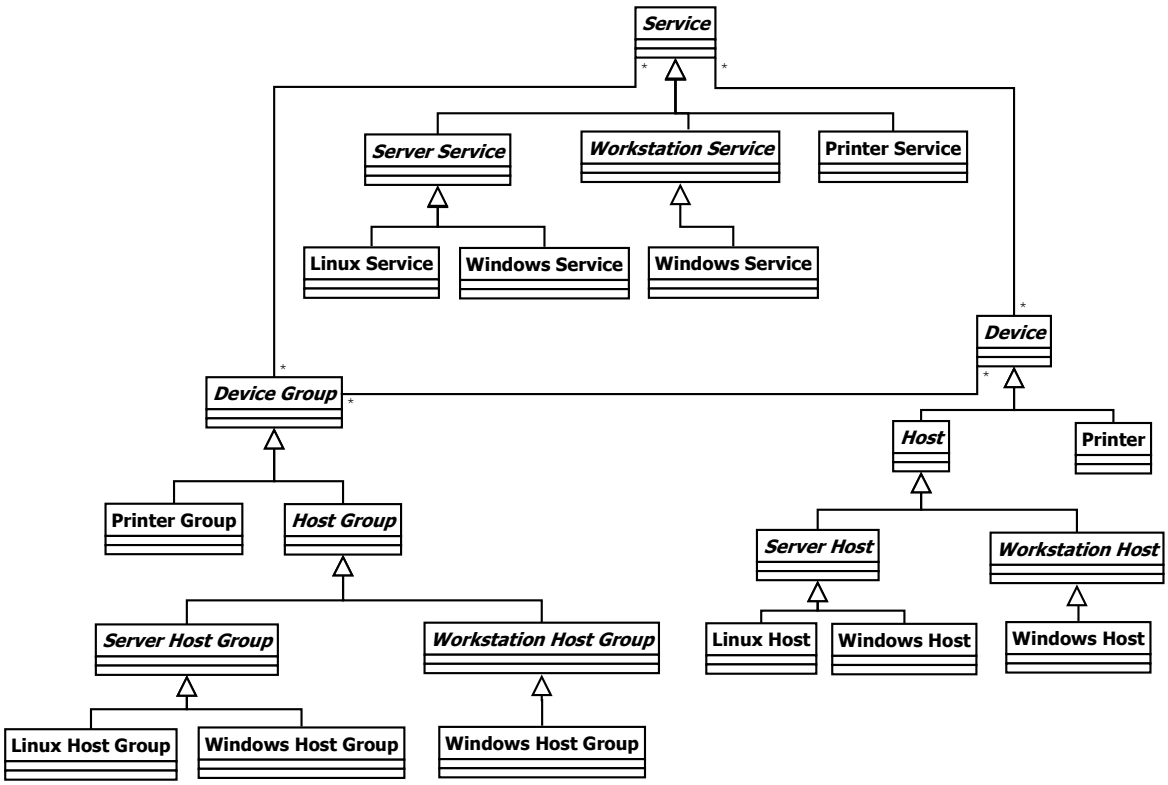

Figure 2.2: Device-Service dependencies

In discussion with the CIP-Pool administrators and other members of the institute who are responsible for this project, a list of CIP-Pool hardware infrastructure information to be monitored has been made as listed in figure 2.3 and 2.4

Host	Checks	Graphs
domain controller & fileserver	Reachability on the CIP-Pool network using ping	Yes
	secure shell service (SSH) status, number of users logged on	-
	samba service status	-
	NIS service status	-
	ZFS health/status	-
printer server	status of the web application	-
	Reachability	Yes
printers	Reachability	Yes
workstations	Reachability	Yes
	Harddisk usage	Yes
	CPU load	Yes
	Memory usage	Yes

Figure 2.3: Host and Checks

Host	Checks	Graphs
server platform	Reachability	Yes
	secure shell service status, number of users logged on	-
	status of the web application	-
	number of zombie process	-
simulation servers	Reachability	Yes
	SSH status, number of users logged on	-
	Hard disk usage and temperature	-
	CPU load, number of running processes, CPU temperature	Yes
	Memory usage (RAM)	Yes
	number of zombie process	Yes
VM servers	Reachability	Yes
	SSH status, number of users logged on	-
	status of the web application	-
	number of zombie process	-

Figure 2.4: Host and Checks

3 System Selection

In order to realise the monitoring system specifications from chapter 2 we proceed as follows. A brief comparison of the monitoring tools that are used in the present days are done. This helps in selecting the tools for this project. Then the architecture and functioning of the selected tools are described.

3.1 State of the art

Network monitoring systems that are predominantly used and which are available as open source software are *Nagios*, *OpenNMS* [3], *Zenoss* [10] and *Zabbix* [9]. These tools are functionally rich, and are constantly developed over time by communities having practical experience with network monitoring.

The important features, which the institute expects from a monitoring system tool, are listed out here.

1. Access control: The tool must support the feature user-level security. This allows an administrator to prevent access to certain parts of the monitoring system based on users/roles.

2. Auto discovery: The tool must support the feature to automatically discover hosts or network devices it is connected to.

3. Data storage method: What data storage methods are supported by the tool to save the network data that is monitored by it?

4. Developer community: How active is the developer community with respect to support and feedbacks.

5. License: *GNU* [18]

6. Logical grouping: The tool supports in organising the hosts or devices it monitors into predefined (logical) groups.

7. Maps: The tool depicts a graphical network map of the hosts and devices it monitors, and their relationships.

8. Plugins: Supports plugins to extend the capabilities of the core tool.

9. *SNMP* [26]: The tool is able to interact with SNMP enabled devices.

10. *Syslogs* [29]: Ability to retrieve system logs.

11. Threshold notifications: The tool is capable of detecting threshold violations and alert the same via email.

12. Trends: The tool must provide trending of network data over a period of time.

13. Web application: A tool with a web-based frontend.

The Network monitoring systems core features are compared in diagram 3.1

Product Name / Features	Nagios	OpenNMS	Zenoss	Zabbix
Access Control	Yes	Yes	Yes	Yes
Auto Discovery	Via plugin	Yes	Yes	Yes
Data storage method	Flatfile, PostgreSQL, MySQL.	PostgreSQL	MySQL	Oracle, MySQL, PostgreSQL
Developer Community	Yes	Yes	Yes	Yes
License	GPL	GPL	GPL	GPL
Logical Grouping	Yes	Yes	Yes	Yes
Maps	Yes	Yes	Yes	Yes
Plugins	Yes	Yes	Yes	Yes
SNMP	Via plugin	Yes	Yes	Yes
Syslog	Via plugin	Yes	Yes	Yes
Threshold notifications	Yes	Yes	Yes	Yes
Trends	Yes	Yes	Yes	Yes
Web application	Yes	Yes	Yes	Yes

Figure 3.1: Feature comparison of network monitoring tools

3.2 System tools - core

In this section we describe all the core tools, i.e. Nagios along with the most important *addon* and the *Plugins* which have been selected for the network monitoring system.

Nagios is a network monitoring tool that gives administrators the ability to examine computers, routers, switches and printers. Nagios is licensed under the terms of the *GNU* General Public License Version 2 as published by the *Free Software Foundation*.

The supervisors of this internship from Department of Statistics and Department for Informatics recommend Nagios as the core monitoring tool. This decision has been taken because all the functional needs as listed in table 2.1 are fullfilled, which is to be seen from the figure 3.1. More over the administrators of the institute have some past working experience with this tool and hence Nagios was chosen.

Nagios system architecture

The Nagios system design is a server/client(agent) architecture. On a network, the Nagios server runs on a host, and the plugins are running on all the remote hosts, whose resources have to be monitored. These plugins send informations to the server, and the server displays these using the web interface.

Nagios system can be divided into three parts

- Daemon: Resides on the server part of the system. It's main function is to schedule checks and process results.
- Plugins: It performs the actual checks on the resource and returns a result to the Nagios server.
- User interface: Nagios displays the results of a check via a web interface.

The error status of a resource check is resolved by Nagios in two ways. A soft state (temporary error state) is resolved when a plugin check returns a warning or an error. This is the first time when Nagios server knows about the error. When this soft alert is raised n times (n is configurable), Nagios marks the resource to a hard state and a hard alert is raised. In this project it is configured that Nagios server sends notifications only when a resource is resolved to a hard state.

The plugins can also be present on the remote system. Nagios *addons NRPE* is designed to allow to execute plugins on remote system.

Nagios *Addons*

addon are software packages that extend Nagios core functionality, in terms of integration with other applications. *NDOUtils addon* has the functionality to persist performance data to relational database like *MySQL* or *PostgreSQL*.

NRPE: This addon is designed to execute Nagios plugins on remote Linux/Unix hosts and this helps Nagios to monitor the local resources of a remote host. These resources are not usually exposed to external hosts, an agent like NRPE must be installed on the remote host. `check_by_ssh` plugin (see page 17) is also used to execute Nagios plugins on remote hosts with help of *SSH*, but it imposes a larger CPU overhead on both the monitoring and remote hosts. It really matters only

when one is monitoring large number of hosts. We opted to NRPE addon because of the lower load it imposes.

Plugins

Plugins are pre-compiled executables or scripts written in arbitrary programming languages to check the status of a host or a service. Nagios uses plugins to check the status of a host on the network.

> *Plugins act as an abstraction layer between the monitoring logic present in the Nagios daemon and the actual services and hosts that are being monitored.* [21]

Such a plugin architecture enables Nagios to monitor almost anything on the network, because it can be easily customised as per needs. Plugins are not distributed with the core Nagios installation package. It can be downloaded from `www.nagiosplug.sourceforge.net/` and `www.nagiosexchange.org/`.

The plugins used in this project are listed along with their functions in table 3.2. Institute's system administrators were involved in the selection to ensure that all the functional needs are met.

Hardware

Plugin name	Functional need
`check_disk`	checks the amount of used disk space on a mounted file system and generates an alert if free space is less than one of the threshold values.
`check_nagios`	checks the status of the Nagios process on the local machine, to make sure the Nagios status log is no older than the number of minutes specified by the expires option. It also checks the process table for a process matching the command argument.
`check_procs`	checks all processes and generates warning or critical states if the specified metric is outside the required threshold ranges. The metric defaults to number of processes. Search filters are applied to limit the processes to check.
`check_swap`	check swap space on local machine
`check_sensors`	check cpu, motherboard and hard drive temperatures
`hotsun_temp.sh`	check cpu temperatures of hosts running on UNIX OS
`check_hddtemp.sh`	check hard drive temperatures
`check_load`	check the current system load average

Threshold alerts

Plugin name	Functional need
`nagios_mail.php`	notification plugin with *Cascading Style Sheets - CSS* [37] formating makes the alerts very legible

Printers

Plugin name	Functional need
`check_hp_print`	check the printer status for low toner, paper jams, door and trays open, offline

Networks

Plugin name	Functional need
`check_by_ssh`	use *SSH* to execute commands on a remote host
`check_http`	used to test the *HTTP* service on the specified host like `amasi`
`check_ping`	check connection statistics for a remote host

OS related

Plugin name	Functional need
`check_nt`	collects data from Windows hosts

Data collection and real time reports

Plugin name	Functional need
`process_perfdata.pl`	used to collect the performance data
`check_pnp_rrds.pl`	checks the performance data collected for *Round-Robin-Database RRD* [11] faults

General

Plugin name	Functional need
`check_users`	checks the number of users currently logged in on a system and generates alarm if the number exceeds the thresholds specified
`check_ssh`	check connection to an specified SSH server and port
`negate`	negates the status of a plugin (returns ok for critical and vice-versa)

3.3 System tools - extensions

In this section we describe all the extention tools, i.e. *Grapher tool* and the *Visualization addon* which are required to extend the functionality of Nagios.

Nagios Event Broker - *NEB*

NEB is an event-driven plug-in framework, which allows modules to register with the event broker, to receive notification of a wide variety of Nagios events and then to act based on those events. This data can be further passed on to an external application for processing it.

Grapher tool

pnp4nagios [4], a grapher tool, is chosen to depict the performance data. This tool can be used to retrieve data from Nagios and can depict real time graphs (*RRD* style graphs) which fullfills the real time reports and graphs generation functional need.

Visualization addon

Reality view must be able to show the monitored information as defined in the previous chapter. We use *NagVis* [2] which is a visualization addon for Nagios. Using data supplied by *NDOUtils* it will update objects placed on a map at certain intervals to reflect the current status. These maps can be customized to ones needs. An example for a map is the physical organization of all hosts in a room or a department. One can also arrange them logically according to functionality like servers/workstations, in order to realise the unified dashboard interface functional need.

Nagios configuration maintainance

A version control system gives us the ability from the very begining of the configuration to manage it, without the risk of losing important changes made to it. We chose *SUBVERSION* [5] as the version control to resolve this functional need. It not only mitigates the risks associated with having multiple authors working on the same configuration data at the same time, but also provides an easy way to have live backups of Nagios configurations. A quick recovery to a stable configuration after some changes have taken place is also ensured.

Summary

Nagios as the core tool along with the plugins and addons builds up the monitoring solution of the institute. One can see that the plugins and addons are main components of Nagios to monitor the network. The next chapter deals with installation of the chosen tools. The configuration and customisation of the tools to realise the monitoring needs as listed in table 2.3 are also described in the next chapter.

4 System Installation and Configuration

The tools installed for this project along with their version number and installation documentation are listed in table 4.1. An important step in this project is to plan and customise the configuration. It is subjective and depends on the implementation scenario of the institute. It is explained in detail in section 4.2 of this chapter.

4.1 Nagios installation

In order to install Nagios, we need a machine running on Linux or Unix OS, and a *C* compiler [23]. The machine should be configured so that it is reachable on the local area network as most of the service checks are performed over the network.

Monitoring system technical specifications

The institute has decided to virtualise the Nagios monitoring system in both test and productive environment. A virtual appliance [35] with *debian* guest OS is hosted on the institute's *VMWARE Server*.

4.2 Nagios configuration

Perform the basic Nagios configuration as in `http://nagios.sourceforge.net/docs/`. The Nagios daemon uses the following configuration files in order to monitor a system.

- Main configuration file: This file contains directives which control the running of Nagios daemon.
- Resource files: This file contains sensitive configuration information like passwords.
- *CGI* configuration file: This file contains directives that affect the operation of the *CGIs*.
- Object definition files: This file contains elements that are involved in the monitoring and notification logic and are called objects. Nagios core installation provides the following sample object templates.

Tool(Version)	Installation Guides
Nagios-3.1.2	http://support.nagios.com/knowledgebase/officialdocs http://www.debianhelp.co.uk/nagiosinstall.htm
NDOUTILS-1.4	http://support.nagios.com/knowledgebase/officialdocs
NRPE-2.x	http://support.nagios.com/knowledgebase/officialdocs
NSClient++	http://nsclient.org/nscp/wiki/doc/installation/manual
NagVis-1.4	http://www.nagvis.org/documentation
PNP4Nagios-0.4.14	http://docs.pnp4nagios.org/
NagiosGrapher-1.7.1	http://www.netways.de/de/produkte/nagios_addons/nagiosgrapher/

Table 4.1: Tool installation source

```
* Hosts
* Host Groups
* Services
* Contacts
* Contact Groups
* Commands
* Time Periods
* Notification
```

In this documentation, we deal with object configuration files, since these files have been been customised to meet the functional needs. This has been done taking into account not only the functional needs, but also maintenance of the configuration files and the growing needs of the institute on a long run. Core Nagios brings in a flexible object-oriented configuration language inherently and this aspect has been harnessed.

We organise the object configuration files as in listing 4.1. This configuration structure has been done as per the relationship model from figure 2.2. As mentioned earlier we keep this directory structure in a version controlled repository.

Templates

Nagios offers templating ability to organize its configuration. In our configuration all hosts and services that are monitored are based on templates in order to avoid redundancy by inheriting the common properties from these templates. The common attributes of the hosts or services which are managed by the monitoring system are

defined once in the template, and each host definition using this template, can add or override attributes that are specific to it. How we have organised this templating is illustrated in listing 4.2.

In listing 4.2 both `gforge` and `amasi` hosts are linux based servers and inherit all common attributes for a linux server from `templates_linux.cfg` which in turn inherits from `templates.cfg` generic host template. We also see that they override

```
|-- commands.cfg
|-- contacts.cfg
|-- printers
|   |-- printer_hosts.cfg
|   |-- printer_service.cfg
|   `-- printer_templates.cfg
|-- servers
|   |-- amasi.cfg
|   |-- cipserv1.cfg
|   |-- cipserv2.cfg
|   |-- cpserver.cfg
|   |-- gforge.cfg
|   |-- hotsun.cfg
|   |-- nagios_server.cfg
|   |-- pythia.cfg
|   `-- servers_hostgroups.cfg
|-- services_linux_base.cfg
|-- services_linux_nrpe.cfg
|-- services_windows.cfg
|-- templates.cfg
|-- templates_linux.cfg
|-- templates_windows.cfg
|-- timeperiods.cfg
`-- workstations
    `-- workstation_hosts.cfg
```

Listing 4.1: Object definition files

`notifications_enabled` property in the respective host definition.

```
###############################################
# TEMPLATES.CFG -  TEMPLATES DEFINITIONS
###############################################

define host
{
name                    generic-host ;Name of this host template
event_handler_enabled   1            ;event handler is enabled
flap_detection_enabled  0            ;Flap detection is enabled
   .......
notifications_enabled   0
register                0
}
```

```
###############################################################
# TEMPLATES_LINUX.CFG - LINUX SERVERS TEMPLATES DEFINITIONS
###############################################################
define host
{
name   linux-server  ;The name of this host template
use   generic-host  ;inherits from the generic-host template
  ..........
}

################################################
# AMASI.CFG -  Host DEFINITION
################################################
define host
{
use             linux-server    ;inherits from the linux-server
                                 template
host_name       amasi
notifications_enabled   1
}

################################################
# GFORGE.CFG -  Host DEFINITION
################################################
define host
{
use             linux-server   ;inherits from the linux-server
                                template
host_name       gforge
notifications_enabled  1
}
```

Listing 4.2: Templates definitions

Host Groups

Host Groups are mainly used to associate groups of devices with groups of related service checks. When a group of one or more hosts offer or share common services, then it would be easier to configure them by putting them under one hostgroup. Listing 4.3 shows the hostgroup configuration for `base-linux-servers` and `http-linux-servers`. The hostgroup `base-linux-servers` are servers running on LINUX and `http-linux-servers` are all linux servers which have *Apache* [17] web server running on it. Linux servers offer *SSH* login service for administration purpose. Hence the functioning of *SSH* application must be monitored on each of them as per the requirements mentioned in page 10. From the service definition in listing 4.4 we can see the advantage of just adding the host group to the *SSH* service. Then all the hosts belonging to this group are then checked by nagios for this particular service. The hosts `amasi, pythia` and `gforge` are the only Linux servers that have *Apache*

webserver running on them. They are grouped into `http-linux-servers` as shown in 4.3. And in the service definition we add the hostgroup.

It is to be noted that any number of hosts can be added to a host group dynamically at any point of time. Further more it help us to regularly check availability and trend reports for all the hosts belonging to a particular host group.

```
################## LINUX HOST GROUP DEFINITION ####################

### Severs which have basic services
define hostgroup{
hostgroup_name  base-linux-servers
alias           Linux Servers
members         pythia,amasi,hotsun,gforge,nagios,
                cipserv1,cipserv2
        }

################## LINUX HOST GROUP HTTP DEFINITION ##############

define hostgroup{
hostgroup_name  http-linux-servers
alias           HTTP Linux Servers
members         amasi,pythia,gforge
        }
```

Listing 4.3: Host group definition

```
################## Service CHECK SSH #################
define service{
.
.
hostgroup_name                  base-linux-servers
service_description             SSH
check_command                   check_ssh
}

################## Service HTTP CHECK #################
define service{
.
.
hostgroup_name                  base-linux-servers
service_description             Apache Webserver Check
check_command                   check_http!-u /htdocs
.
.
}
```

Listing 4.4: Service definition Linux servers

The following hostgroups were configured for this project. The concept for the configuration is similiar as shown in listing 4.3

- cip-printers: hostgroup for network printers
- windows-workstations: hostgroup for clients running on Windows
- nrpe-linux-servers: hostgroup for servers running on Linux
- windows-servers: hostgroup for servers running on Windows
- nrpe-linux-servers-vhosts: hostgroup for clients running on Linux with *NRPE* daemon and are *VHosts*

Hosts

All hosts which are monitored are members of a one or more hostgroups. This design helps us to add any number of host dynamically at any point of time in future without much changes to the configuration files. One could define the host configuration in a single file. In order to change the configuration of a particular host, one needs to have a good overview, and hence we have split the host configuration into many files, i.e based on its function in the CIP-Pool network. The configuration files are organised as in listing 4.5.

One of the host definition is as in listing 4.6. The other host configuration follows the same pattern.

```
|-- printers
|   |-- printer_hosts.cfg
|-- servers
|   |-- amasi.cfg
|   |-- cipserv1.cfg
|   |-- cipserv2.cfg
|   |-- cpserver.cfg
|   |-- gforge.cfg
|   |-- hotsun.cfg
|   |-- nagios_server.cfg
|   |-- pythia.cfg
|-- workstations
    `-- workstation_hosts.cfg
```

Listing 4.5: Host object definition files

```
define host{
use             windows-client  ; Inherit values from a template
host_name       cip01
alias           Windows Client
address         cip01.stat.uni-muenchen.de
}
```

Listing 4.6: Host definition Windows Client

In listing 4.6 the directive `address` is used to define the address of a host. Normally, this is an IP address. It is used to check the status of the host. As already mentioned, when the CIP infrastructure of institute was built up, all the servers and printers were configured with a static IP and use the *DNS* service offered by *LRZ* to map the host name to the IP.
However the workstations receive dynamically their IPs, upon request, from the *DHCP* server managed by *LRZ*. The institute has no intention to install and maintain a *DNS* server of its own at this point of time. This is being mentioned here because nagios host adressing for the clients receiving IPs through *DHCP* is not forseen. The `address` directive entry should suffice to address the host else the configuration fails to perform the service checks. Institute uses *WINS* to resolve hostnames of the workstations. But this only works under Windows. Since NAGIOS server runs on Linux, we had to solve this problem at this stage before we proceed.
One ideal solution is to have *DNS* server and in addition either configure the *DHCP* server to offer static IPs to the clients. Since the institute seeks another solution, a *Perl* script [8] was written as in listing 4.7. It is set up as a *Cron* job to retrive the hostname and IP mapping from the *WINS* server and write it into `/etc/hosts` file of the NAGIOS host. In this way the Nagios server can reach the hostnames specified in its host definition.

```
#!/usr/bin/perl
my $winsHostname;
my $winsServer = "10.153.53.101";
my $resultFile="/etc/hosts";

#This loop handles the first fifty cip* hostnames
for (my $count=1; $count<50; $count++){
  if ( $count < 10 ){
    $winsHostname = "cip0".$count;
  } else {
    $winsHostname = "cip".$count;
  }

  my $nmbResult;
  $nmbResult = `nmblookup -U $winsServer -R $winsHostname
               | grep -v $winsServer`;

  if ( $nmbResult =~ /name_query/ ){
    #This means that no host with this hostname
      was found on the WINS server.
    #remove the line containing the given hostname,
     because it is invalid.
    `perl -i -n -e 'print if not /$winsHostname/i' $resultFile`;
  } else {

    #We continue with parsing the ip address - result looks like:
     "10.153.53.72 cip26<00>"
    my $ip =  substr $nmbResult, 0, index($nmbResult, ' ');

     #remove the line containing the given hostname,
      because it might be not actual.
```

```
    `perl -i -n -e 'print if not /$winsHostname/i' $resultFile`;

    `echo "$ip $winsHostname" >> $resultFile`;

    #print "$ip $winsHostname\n";
  }
}
exit 0;
```

Listing 4.7: Resolve *WINS* adresses *Perl* script

In brief this script begins with setting the configurations required for its execution. *WINS* server IP is defined and also the file where the result has to be written into. `nmblookup`, a linux command uses the *NetBIOS* names (cip workstations host names running under windows) to query and map them to their IP addresses. Finally we parse the output and write it in the result file.
The script also takes care that stale informations, i.e. the old IPs are regularly removed from result file and hence keeping it up to date for the proper functioning of Nagios. It has to be mentioned at this point that this script is not an alternative solution but only a workaround! Some of the known disadvantages are that if the windows clients are not running then they do not have an entry in the *WINS* server and therefore Nagios plugins resolve it as an unknown host and not as unreachable.
One more point is when *WINS* server fails to function, then most of the service checks will again fail because the plugins will be unable to resolve the host name. But this is solved by monitoring the *WINS* server itself. The system administrators are notified and will be able to react accordingly in such a situation. Institute has decide to use this workaround at present.

Services

In short services are the central objects in the monitoring logic. They are concrete services which are associated with a device or a host and are either attributes of a device or a host like CPU load, disk usage and so on. It could also be the actual services provided by a host like webserver or network login through *SSH*.
In our case there exist some services as mentioned in table 2.3 that are offered by multiple hosts. In order to keep the configuration and modification of the service definition simple we create a single service definition. And in the `hostgroup_name` directive we specify the name of one or more hostgroups for which this service should be created. From the `hostgroup` explanation we know that this definition would map this service on all hosts that are members of hostgroups.
A service definition which has been created in this project is shown in listing 4.8. It checks the number of currently running process on all linux servers. This is achieved by including the `nrpe-linux-servers` and `nrpe-linux-servers-vhosts` (multiple hostgroups).
The templating mechanism mentioned in listing 4.2 holds good for a service definition too and is used also in a similar way as mention before.

```
#######################################################
#check the number of currently running procs on the host
#######################################################
define service{
use                     generic-service
hostgroup_name          nrpe-linux-servers,
                        nrpe-linux-servers-vhosts
service_description     Total Processes
check_command           check_procs!10!20!CPU
}
```

Listing 4.8: Service definition

A service uses the `check_command` directive to tell Nagios deamon which plugin, programs or scripts it should execute to perform the actual service. In the configuration shown, the plugin `check_procs` is used to execute the service and it is fed with 3 parameters, which actually means alert if CPU usage of any processes raises above 10% or 20%. All the services definitions have been configured as per requisite of the institute as listed in table 2.3. The screenshots 4.1, 4.2 and 4.3 show the same. The service definition files are as shown in listing 4.9.

```
|-- printers
|   |-- printer_service.cfg
|-- servers
|   |-- amasi.cfg
|   |-- gforge.cfg
|   |-- hotsun.cfg
|   |-- nagios_server.cfg
|   |-- pythia.cfg
|-- services_linux_base.cfg
|-- services_linux_nrpe.cfg
|-- services_windows.cfg
```

Listing 4.9: Service definition files

Notifications

Designing notification rules are also part of Nagios configuration. It involves extracting relevant information and notifying the concerned people without any delay. It involves in customizing notifcations as well as setting priority, so that the concerned admin can react to the email accordingly. The need of the institute in this aspect is quite simple and straightforward. Email notification has been setup as per the requirement of the institute. In short Nagios notifies according to the status defined in the plugins. In our case a notification takes place in case a host or a service attains a critical status. We use an additional plugin and not the standard Nagios email plugin. This plugin `nagios_mail.php` can send emails with *CSS*.

Host	Service	Status	Status Information
pythia	CPU Statistics	WARNING	CPU STATISTICS WARNING : user=84.42% system=9.55%, iowait=0.00%, idle=6.03%, nice=0.00%, steal=0.00%
	CPU Temperature	OK	k8temp-pci-00c3 Adapter: PCI adapter Core0 Temp: +47.0Â°C Core1 Temp: +45.0Â°C
	CPU load per cpu	CRITICAL	CRITICAL: Used CPU CRITICAL: CPU0 100.00% CPU1 100.00%
	Check Network card	OK	eth0: negotiated 1000baseT-FD flow-control, link ok
	Current Load	OK	OK - load average: 2.44, 2.73, 2.78
	Current Users	OK	USERS OK - 0 users currently logged in
	HDD Temperature	OK	OK: Temperature is below warn treshold (/dev/sda is 38)
	PING	OK	PING OK - Packet loss = 0%, RTA = 22.02 ms
	Root Partition	WARNING	DISK WARNING - free space: / 18162 MB (13% inode=99%):
	SSH	OK	SSH OK - OpenSSH_5.1p1 Debian-7 (protocol 2.0)
	Swap Usage	OK	SWAP OK - 100% free (6236 MB out of 6236 MB)
	Total Processes	OK	PROCS OK: 76 processes
	VMSever WEB-APP on Pythia	OK	HTTP OK HTTP/1.1 200 OK - 1936 bytes in 0.002 seconds
	Zombie Processes	OK	PROCS OK: 0 processes with STATE = Z

Figure 4.1: The Linux host *pythia* with its services

Host	Service	Status	Status Information
cip05	C:\ Drive Space	OK	c: - total: 39,06 Gb - used: 18,13 Gb (46%) - free 20,93 Gb (54%)
	CPU Load	OK	CPU Load 0% (5 min average)
	Explorer	OK	Explorer.EXE: Running
	Memory Usage	OK	Memory usage: total:4956,32 Mb - used: 998,00 Mb (20%) - free: 3958,32 Mb (80%)
	NSClient++ Version	OK	NSClient++ 0.3.6.540 2009-03-26
	Uptime	OK	System Uptime - 2 day(s) 8 hour(s) 43 minute(s)

Figure 4.2: A Windows host with its services

Contacts, Contact Groups

Contact groups help us to realise role based notification for different groups. This is done by adding an user to one or more contact group using the contactgroups attribute of the contact object, or to associate a contact from the contact group itself by enumerating them in the members attribute. Our configuration for the contact and contact group are as shown in listing 4.10.

```
#####################CONTACT GROUPS#########################

define contactgroup{
        contactgroup_name       nagios_admins
        alias                   Nagios Administrators
        members                 kunja,manuel
        }
######################     CONTACTS   ####################
```

```
define contact
{
contact_name        manuel
...
email               manuel@example.com
}
```

Listing 4.10: Contact definition

Host	Service	Status	Status Information
DruckerE03	ALERTS	WARNING	Warning - DRUCKERWARTUNG DURCHFÜHREN(5 4 10 5 99999 10)
	COUNTER	OK	OK - Total: 200996
	HARDWARE	WARNING	Warning - HP LaserJet 4100 Series Status=3
	PAPER	OK	OK - TRAY 2
	PARTS	OK	OK - DRUCKERABDECKUNG Status=4
	TONER	CRITICAL	Critical - TONERPATRONE HP C8061A 0% (0 of 4500 left)

Figure 4.3: *Drucker E03* with its services

Time Periods

User objects have the `service_notification_period` and `host_notification_period` to limit when a notifications can occur. Time periods can use other time periods as exclusions to limit the range of the time period. Hosts and services have time periods associated with them that limit when host checks are performed (`check_period`) and when notifications are sent (`notification_period`). Nagios separates the check periods and notification periods. In the listing 4.11 a service is checked 24x7 and triggers notifications also in this time period.

```
check_period    24x7

notification_period 24x7

define timeperiod{
        timeperiod_name 24x7
        alias           24 Hours A Day, 7 Days A Week
        sunday          00:00-24:00
                .......
        }
```

Listing 4.11: Time Period definition

4.2.1 Addons and grapher configuration

In this subsection all the configuration about the grapher tool and the addons are described.

NDOutils

The basic configuration should be performed as in `http://nagios.sourceforge.net/docs/ndoutils/NDOUtils.pdf`. The *NDO Utils* package captures service, host configuration information and host service events from Nagios and populates a database with these informations. Since the institute has only one instance of nagios monitoring the whole landscape, a single server, single instance configuration setup is chosen. This is explained well in the *NDOUtils* documentation.

NSClient++

The basic configuration should be performed as in `http://nsclient.org/nscp/wiki/doc/configuration`.

The only configuration file is `NSC.ini`. Here it is configured to set the `allowed_hosts` derective to the IP of Nagios. All other directives were set to standard values.

Grapher

The basic configuration should be performed as in `http://docs.pnp4nagios.org/pnp-0.4/start`. Planing the *pnp4nagios* grapher means to select the mode in which one gets data from Nagios into *PNP*.

Default mode has been configured in this project to directly call the *PNP RRD* file creation script (`process_perfdata.pl`) with performance data from Nagios every 300 seconds.

Performance data can be data containing multiple status returned by a check. Such result helps one to separate data for depiction. For example a newtork check can result in data useful for administrators vs.data which is useful for students. From figure 4.4 we see an example of performance data gathered in this project setup.

Nagvis Configuration

NagVis configuration planing involves in creating maps that show the status of servers and workstations and make it visually very easy for staff to detect problems. A photo of each CIP room is the background for the map.
The overview of all the maps are used for the start page, which has an icon for each location monitored that links to a detailed NagVis map specifically for that location. NagVis is enabled to evaluate only hard states for routine work. It turns out to be quite useful if not every temporary soft critical state immediately generates a critical warning

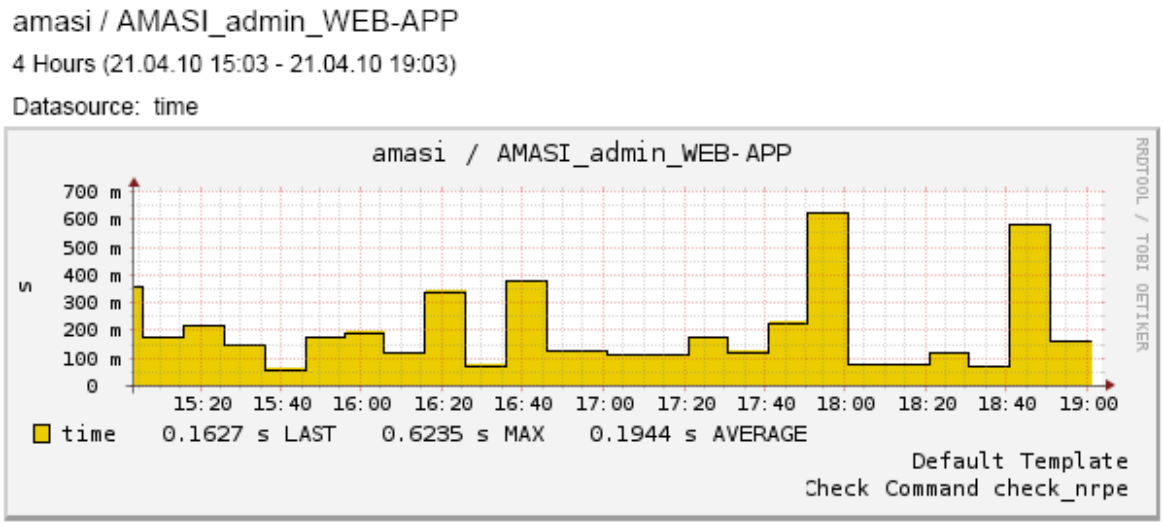

Figure 4.4: Performance data graph for administrators

symbol. All the maps are configured under `/$NAGIOS_HOME/share/nagvis/etc/maps` and are organised as in listing 4.12.

```
|-- E01-A.cfg
|-- E01-B.cfg
|-- E02.cfg
|-- E03.cfg
|-- E06.cfg
|-- E09.cfg
|-- E10.cfg
|-- autobackup.status
```

Listing 4.12: *Nagvis* map files

In conclusion, Nagios configuration of the institute is wholly based in terms of templates. We use naming conventions for hosts and services well known within the institute. These conventions help the administrators to identify the location of a faulty host very easily. Using these techniques we have reduced the size and number of our configuration files.

5 Scope for Improvement and Summary

There is some further scope for improvement. The current implementation acts as a good base for the addition of new features and modules. In this chapter we list out a few points.

Network hierarchy

In the official Nagios documentation `http://nagios.sourceforge.net/download/contrib/documentation/`, the section "Reachability of Network Hosts" deals with how Nagios monitor services on local and remote hosts. In this project the configurations related to this were simple and straightforward as it was all on a local network of CIP-Pool. Hence all the hosts with respect to Nagios were set on a flat hierarchy. However if the institute desires to expand its monitoring capability to other network then the cause of network outages cannot be precisely met with the present set up. This can be easily done by defining parent/child relationships as mentioned in the documentation.

Manageable switches

In this project it was not possible to monitor the status of network switches. The institute at present has unmanaged switches. According to the official Nagios documentation there is no way to monitor such switches. If in the future manageable switches become part of the CIP infrastructure, then they can be monitored i.e. via *SNMP* to query status information.

Auto discovery

A very useful feature for a monitoring tool is to detect hosts in the network automatically. This feature was not used by us in this project because institute wanted to configure all the hosts manually for the first time, so that they have a better control and overview of configuration. Once this has been done it is very handy to have this feature for new hosts, since this makes the configuration very easy. This feature is not available with Nagios core, but can be integrated as a plugin into Nagios. For further details please see the nagios Addons documentation [6].

***WAP* domain**

Nagios configuration as explained in chapter 4 has been done using templating mechanism. This easily accommodates monitoring of more network instances. It is planned

in the near future to widen the present Nagios monitoring to include the *WAP* domain too.

Summary

An open source tool like Nagios for monitoring provides a platform to collect and display available data and system information based on the administrators preference. When combined with other plugins, addons and grapher tools, long term performance data and information useful for trouble shooting network problems can be made available.

In this project we have deployed a network monitoring product Nagios for the institute with salient features as listed in table 2.1. Nagios being able to monitor all the services listed out in table 2.3, proves to be a good monitoring solution and provides a basis for extending it to other domains of the institute.

I would like to extend my special thanks to my supervisors, Dr. Nils gentschen Felde and Manuel J. A. Eugster for having supported me throughout this project.

List of Figures

Listings

Bibliography

[1] Domain name system. http://tools.ietf.org/html/rfc5395, 2008.

[2] Nagvis. http://www.nagvis.org/, 2008.

[3] Opennms. http://www.opennms.org/, 2008.

[4] pnp4nagios. http://docs.pnp4nagios.org/pnp-0.4/start, 2009.

[5] Subversion. http://subversion.tigris.org/, 2009.

[6] Nagios autodiscovery. http://exchange.nagios.org/directory/Addons/Configuration/Auto%252DDiscovery, 2010.

[7] Nagiosgrapher. https://www.nagiosforge.org//gf/project/nagiosgrapher/, 2010.

[8] The perl programming language. http://www.perl.org/, 2010.

[9] Zabbix. http://www.zabbix.com/, 2010.

[10] Zenoss. http://www.zenoss.com/, 2010.

[11] OETIKER+PARTNER AG. Rrdtool. http://oss.oetiker.ch/rrdtool/, 2009.

[12] W. Barth. *Nagios: System and network monitoring.* No Starch Press San Francisco, CA, USA, 2008.

[13] D. Bennett, M. Schubert, J. Gines, A. Hay, and J. Strand. *Nagios 3 Enterprise Network Monitoring: Including Plug-Ins and Hardware Devices.* Syngress Publishing, 2008.

[14] Microsoft Corporation. Wins:technical reference. http://technet.microsoft.com/en-us/library/cc736411%28WS.10%29.aspx, 2010.

[15] DFG e.V. Computer-investitions-programm. http://web.archive.org/web/20060519013854/http://www.dfg.de/forschungsfoerderung/wissenschaftliche_infrastruktur/wgi/geraete_im_hbfg_verfahren/was_ist_hbfg/cip.html, 2002-6.

[16] DFG e.V. Arbeitsplatzrechnern fuer wissenschaftler. `http://www.dfg.de/download/programme/grossgeraete_der_laender/allgemein/21_11/21_11.pdf`, 2010.

[17] The Apache Software Foundation. Apache http server documentation. `http://httpd.apache.org/docs/`, 2009.

[18] Inc. Free Software Foundation. Gnu. `http://www.gnu.org/licenses/licenses.html`, 2010.

[19] E. Galstad. Nagios. *Open source host, service and network monitoring program*, 2008.

[20] Ethan Galstad. Nagios: Comprehensive it infrastructure monitoring. `http://www.nagios.org/`.

[21] Ethan Galstad. Nagios: Documentation. `http://nagios.sourceforge.net/docs/nagios-3.pdf`, 1999-2007.

[22] Ethan Galstad. Nagios: Nrpe documentation. `http://nagios.sourceforge.net/docs/nrpe/NRPE.pdf`, 1999-2007.

[23] GNU GCC. the GNU Compiler Collection.

[24] GFORGE. Gforge: Home page. `http://gforge.org/gf/`, 2010.

[25] Control Systems GmbH. Pcounter software. `http://www.pcounter-germany.com/`, 2010.

[26] Martin Lee Schoffstall Jeffrey D. Case, Mark Fedor and James R. Davin. Snmp: Rfc1157. `http://www.faqs.org/rfcs/rfc1157.html`, 1990.

[27] D. Josephsen. *Building a Monitoring Infrastructure with Nagios*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2007.

[28] Thorsten Kukuk. Linux nis/nis+ projects. `http://www.linux-nis.org/`, 2007.

[29] Chris Lonvick. Ietf. `http://www.employees.org/~lonvick/index.shtml`, 2010.

[30] SAS. Sas: Home page. `http://www.sas.com`, 2010.

[31] T. Scherbaum. *Praxisbuch Nagios*. O'Reilly, 2009.

[32] SPSS. Spss: Home page. `http://www.spss.com`, 2010.

[33] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1998.

[34] Samba Team. samba.org. `http://www.samba.org`, 2010.

[35] vmware. Virtual appliances. `http://www.vmware.com/appliances/`, 2010.

[36] Inc VMware. Vmware. `http://www.vmware.com/de/products/server/`, 2010.

[37] W3C. Cascading style sheets. `http://www.w3.org/Style/CSS/`, 2010.

[38] W3C. Common gateway interface. `http://www.w3.org/CGI/`, 2010.