Munin - Benachrichtigungsdienst und graphische Visualisierung von Netzwerk- und Serviceüberwachunginformationen

STUDIENARBEIT II

für die Prüfung zum
Bachelor of Engineering

des Studienganges

Informationstechnik

an der Dualen Hochschule Karlsruhe

von

Andreas Paul

Bearbeitungszeitraum:	09.02.2009 - 10.05.2009
Matrikelnummer:	108467
Kurs:	TIT06GR
Theoriesemester:	6
Ausbildungsfirma:	Forschungszentrum Karlsruhe GmbH (FZK) Steinbuch Centre for Computing Hermann-von-Helmholtz-Platz 1 76344 Eggenstein-Leopoldshafen
Betreuer:	Herr Ralf Brune

Inhalt

Dort thront Odin, der höchste Gott, den die Menschen auch Wodan nennen, in Walhalla, der größten und prächtigsten Halle, und waltet über der Welt und über den Menschen. Auf seinen Schultern sitzen zwei Raben, Hugin, der Gedanke, und Munin, das Gedächtnis, die auf sein Geheiß täglich ausfliegen, und raunen ihm ins Ohr, was sie gesehen und gehört haben.

Ernst Probst - „Nordische Göttersagen“ [Pro02] Seite 16

1 Einleitung

Munin ist ein Monitoring-Tool, welches Computer, Services und Ressourcen überwacht und die entsprechenden Daten abspeichert. Alle Informationen werden in graphischen Schaubildern durch ein Webinterface dargestellt.
Als Überwachungsmechanismus setzt Munin auf Schwellwerte, die durch den Benutzer frei anpassbar sind. Werden diese Werte über- bzw. unterschritten gibt Munin eine Warnung aus oder schlägt bei einer kritischen Überschreitung entsprechend Alarm.
Im Gegensatz zu den meisten Konkurrenzprodukten legten die Entwickler von Munin den Schwerpunkt auf einen simplen Aufbau bzw. eine schnelle und unkomplizierte Inbetriebnahme der Anwendung. Bereits nach der Installation stehen eine hohe Anzahl von Plugins zur Verfügung. Doch auch eigene Munin-Skripte lassen sich aufgrund des simplen Aufbaus schnell und unkompliziert selbst programmieren.
Das Munin mehr Wert auf die Visualisierung der Überwachungsdaten legt, als auf eine komplexe Überwachung- oder Alarmierungslogik, stellt sich im direkten Vergleich mit Nagios oder bereits durch das schlicht gehaltene Webinterface heraus. Dafür lässt sich Munin recht einfach in ein bereits vorhandenes Nagios-Überwachungssystem integrieren und sich somit die Vorteile beider Anwendungen nutzen.

2 Aufgabenstellung

Die Netzwerkstruktur und Serverlandschaft der Dualen Hochschule Karlsruhe soll mit Hilfe einer Softwareanwendung überwacht werden. Dies wurde bereits von Christian Lang im vorherigen Praxissemester mit dem „Monitoring-Tool“ Nagios realisiert. Nun soll Munin als weitere mögliche Anwendung für die Überwachung getestet und verglichen werden.

Dafür soll im Laufe dieser Arbeit die Anwendung in einer Testumgebung installiert und auf die angebotenen Features und Möglichkeiten hin untersucht werden. Dabei wird sich mit einer Gegenüberstellung zwischen Nagios und Munin auseinandergesetzt und die Möglichkeit der Zusammenarbeit dieser zwei Anwendungen eruiert.

3 Architektur

Munin verwendet eine sogenannte Master-Node-Architektur, siehe Abbildung 1. Hierbei emittelt jeder Rechner seine Messwerte selbst und der Master holt sich diese Daten mittels diverser Agenten, den sogenannten Munin-Nodes, ab. Deshalb wird der Master lediglich zur Verarbeitung der Überwachungsdaten genutzt.

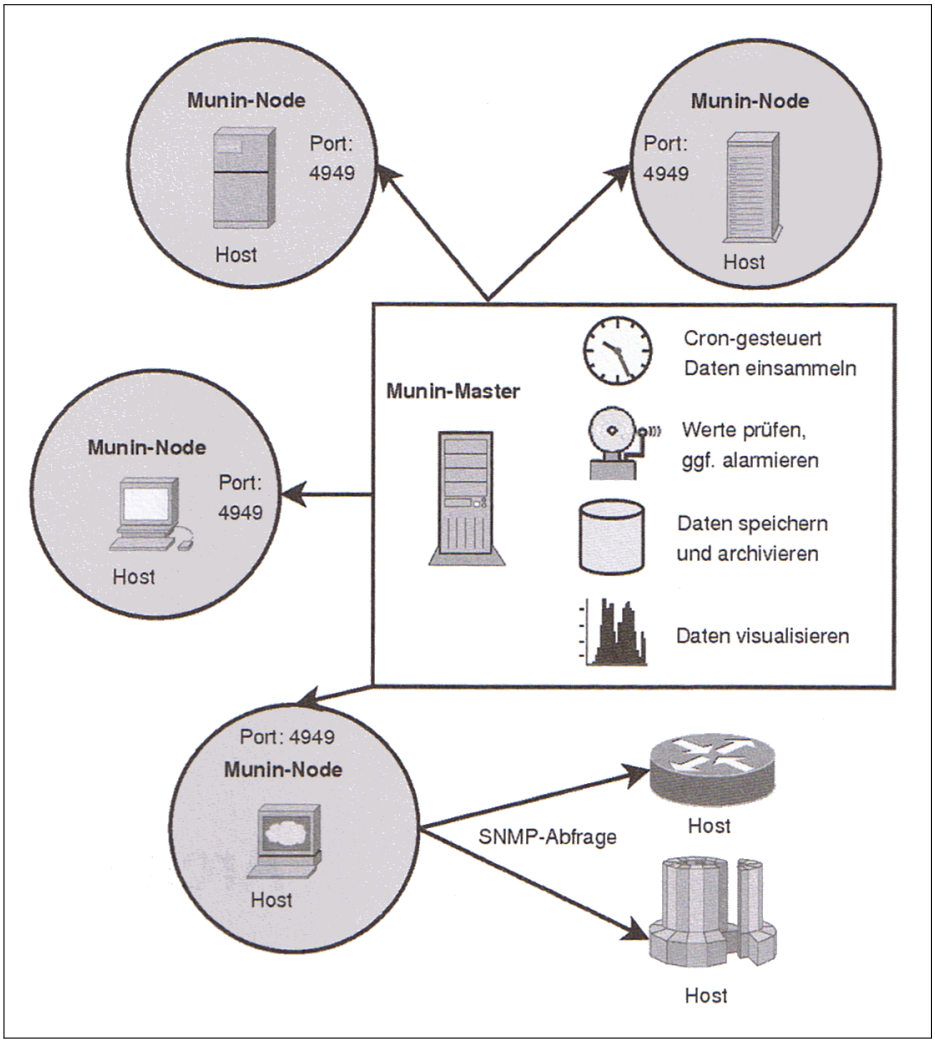

Abbildung 1: Munin Master-Node-Konzept[1]

[1]Quelle: [Munin08] S. 20

Der Munin-Master ist kein Daemon, sondern besteht aus Skripten, die in regelmäßigen Zeitabstanden als Cron-Job ablaufen:

- „`munin-update`“ dient zum Abrufen der Messwerte bei den zu überwachenden Munin-Nodes. Hierbei wird auch die Datenbank aktualisiert oder ggf. erzeugt.
- „`munin-limits`“ vergleicht die Messwerte mit ggf. vorgegebenen Schwellwerten und versendet bei Bedarf Benachrichtigungen, wenn Werte das Warnlevel überschreiten, in kritische Bereich gelangen oder wenn Entwarnung gegeben werden kann.
- „`munin-graph`“ erzeugt die Munin-Graphen.
- „`munin-html`“ dient zum Erstellen der HTML-Seiten der Munin-Übersicht.

Standardmäßig werden diese Skripte im fünf Minuten Rythmus aufgerufen. Dabei baut der Munin-Master viele parallele - im Ausgangszustand unverschlüsselte - TCP-Verbindungen zu den diversen Node-Hosts auf.
Zusätzlich wird für den Betrieb des Munin-Masters auf einem Rechner folgende vorab konfigurierte Software vorausgesetzt:

- einen Webserver, der Zugang zu den Graphen verschafft.
- ein Programm mit dem sich die Warnmeldungen versenden lassen. Beispielsweise einen SMTP-Server oder einen Nagios Service Check Acceptor (NSCA), der dafür sorgt, dass der Nagios-Server Alarm schlägt.
- Werkzeuge und Bibliotheken des *RRDtool*-Projekts[2] zur Speicherung der Daten und zum Zeichnen der Munin-Graphen.

[2] http://oss.oetiker.ch/rrdtool/

3.1 Einsammeln der Daten

Das periodisch ausgeführte Perlskript „`munin-update`“ kümmert sich um das Abholen neuer Messwerte von den Nodes. Dazu wird als Erstes die Datei „`munin.conf`“ geparst, um die zu überwachenden Nodes zu ermitteln.

Ein Eintrag für einen Node-Host in dieser Datei sieht folgendermaßen aus:

```
1 [munin.example.net]
2         address                         localhost
3         port                            6088
4         df.warning                      20
5         df.critical                     10
6         contacts                        paul
7         ping_unilabad.contacts          lang
```

Listing 1: Beispielhafte Definition eines Munin-Nodes

Der String in den eckigen Klammern wird als Identifikationsnamen für den Host verwendet und dieser Node wird auch im Webinterface unter diesem Namen aufgelistet.

Das Attribut *address* gibt die IP-Adresse des zu überwachenden Hosts an und mit *port* kann ein vom Standardport (4949) abweichender Port angegeben werden.

Die einzelnen Schwellwerte der verschiedenen Plugins werden auch in diesem Eintrag angegeben, wenn sie von den vorkonfigurierten Werten abweichen sollen. Dafür wird der Name des Plugins mit den Suffixen *.warning* und *.critical* für die jeweiligen Schwellwerte gesetzt. Im obigen Beispiel wird „`df`“ für den freien Festplattenspeicherplatz mit dem Schwellwert von 20% für Warnungen und 10% als kritischen Wert für den Host definiert.

Unter dem Attribut *contacts* werden die Kontaktnamen angegeben, die bei einer Überschreitung eines Schwellwerteres kontaktiert werden sollen. Hierbei gibt es zu sagen, dass diese Überschreitung bei egal welchem Plugin für die Benachrichtigung dieser Kontakte führt.

Soll ein Kontakt nur bei einem bestimmten Plugin benachrichtigt werden, muss es analog zu der Schwellwertdefinition mit dem Pluginnamen und dem Suffix *.contacts* explizit angegeben werden. Hier wird der Kontakt *lang* nur bei einer Überschreitung des Plugins *ping_unilabad* benachrichtigt. Bei diesem Plugin wird der Host *unilabad* mit einem Ping überwacht. Für weitere Details zu dieser Art von Plugin siehe Kapitel 3.3.1.
Weiterhin lässt sich einstellen, dass ein Kontakt generell nur bei kritischen Werten benachrichtigt werden soll. Die Kontakte müssen zuvor in der „`munin.conf`“ definiert werden.
Nach der Ermittlung der Nodes werden die Hosts in der Regel parallel nach den neuen Messwerten abgefragt. Hierfür erzeugt Munin für jeden in der Konfigurationsdatei angegebenen Rechnern einen eigenen Prozess. Das parallele Abarbeiten hat den Vorteil, dass die Abfrage nicht bis zum Timeout hängen bleibt, wenn ein einzelner Munin-Node nicht erreichbar ist. Jedoch verschlingt das Erzeugen eines eigenen Prozesses für jeden Node bei vielen zu überwachenden Rechnern viele Ressourcen, so dass bei größeren Umgebungen viel Arbeitsspeicher und schnelle Multicoreprozessoren unabdingbar sind.
Ein beispielhafte Ausführung eines Munin-Plugins für die Plattenbelegung gibt folgende Werte zurück:

```
paul@iwrpaul:/etc/munin/plugins$ ./df
_dev_sda2.value 23
tmpfs.value 0
varrun.value 1
varlock.value 0
udev.value 1
tmpfs.value 1
lrm.value 1
_dev_sda5.value 15
```

Abbildung 2: Beispielhafte Ausführung eines Munin-Plugins

Wenn man diese Werte mit den realen Werten vergleicht, erkennt man, dass das Munin-Plugin einfach die Prozentwerte des verbrauchten Speicherplatzes als Wert verwendet.

```
paul@iwrpaul:~$ df
Filesystem           Size  Used Avail Use% Mounted on
/dev/sda2             28G  6.2G   22G  23% /
tmpfs                505M     0  505M   0% /lib/init/rw
varrun               505M  132K  505M   1% /var/run
varlock              505M     0  505M   0% /var/lock
udev                 505M  2.8M  502M   1% /dev
tmpfs                505M   12K  505M   1% /dev/shm
lrm                  505M  2.0M  503M   1% /lib/modules/2.6.27-14-generic/volatile
/dev/sda5             46G  6.4G   38G  15% /home
```

Abbildung 3: Reale Werte des Systemtools *df*

Aus diesen Werten erstellt „`munin-graph`“ automatisch folgenden Graphen:

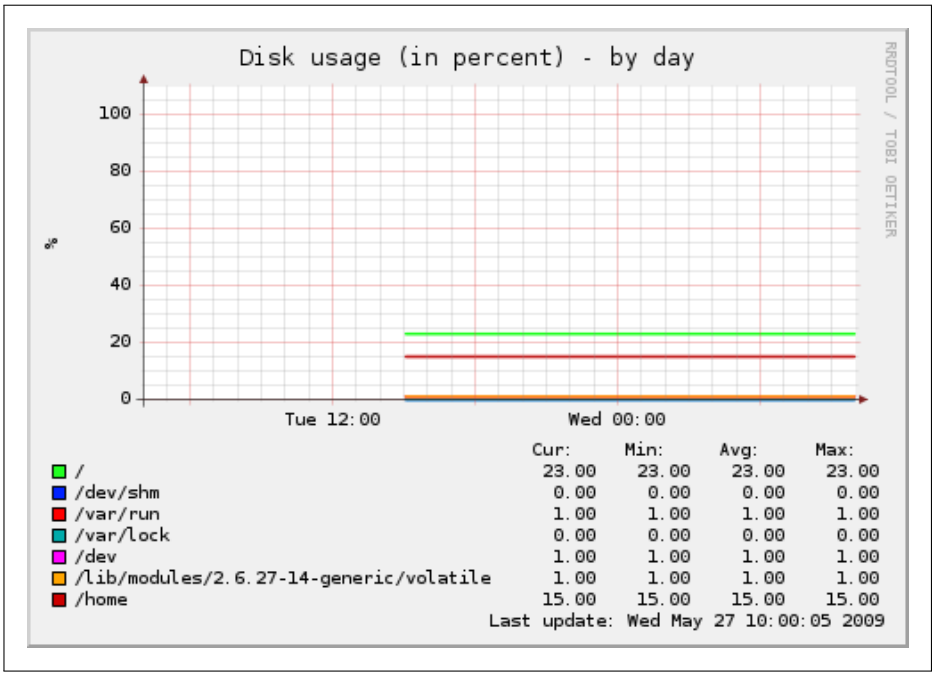

Abbildung 4: Visualisierung der Werte des Munin-Plugins *df*

Die über das Netzwerk ermittelten Werte landen nach entsprechender Bearbeitung in RRD-Dateien. Im Skript „`munin-update`“ ist fest eingetragen, welche zeitliche Auflösung die Datenbasis und dadurch auch die Munin-Graphen aufweisen.

Das bedeutet, dass für einen Messpunkt in der Wochenübersicht 30 Minuten - also sechs Messwerte - benötigt werden und für die Monatübersicht werden bereits 48 Messpunkte bzw. zwei Stunden benötigt. Siehe hierfür Tabelle 1.

Alter der Daten	**Auflösung**
bis zu 30 Stunden	5 Minuten
bis zu 9 Tagen	30 Minuten
bis zu 45 Tagen	2 Stunden
bis zu 450 Tagen	1 Tag

Tabelle 1: Zeitliche Auflösung der Datenbasis[3]

In der Abbildung 5 konnte der „`munin-node`“ noch keinen kompletten Tag Messwerte sammeln, weshalb die Jahresübersicht noch keine Messwerte liefern kann und nur mit dem Platzhalterwert *nan* (für nicht defniert) aufgefüllt ist.

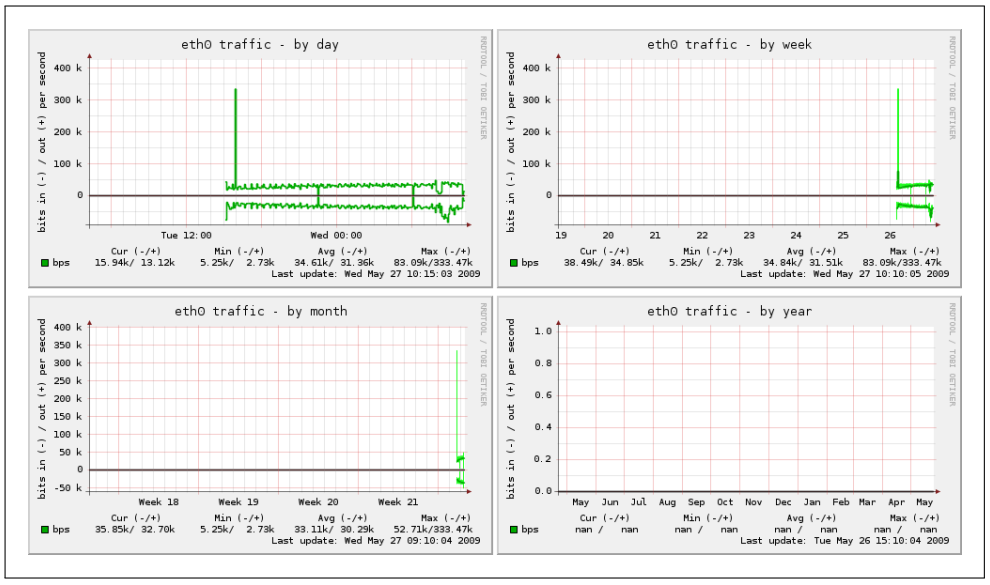

Abbildung 5: Visualisierung der Messwerte in verschiedenen Zeitauflösungen

[3]Quelle: [Munin08] S. 24

3.2 Round-Robin-Datenbanken

Round-Robin-Datenbanken sind nicht mit bekannten Datenbanksystemen vergleichbar, da sie in einem proprietären, binären Dateiformat vorliegen. Daher scheidet der Zugriff per SQL, Texteditor oder über einen Netzwerkport aus. Um auf Round-Robin-Datenbanken zuzugreifen müssen die dafür entwickelten RRD-Tools verwendet werden. Diese nehmen die aktuellen Messwerte entgegen und schreiben für jeden Messinterval einen Wert in eine Binärdatei. Aus den gesammelten Datenbestand können dann die Werte visualisiert oder Statistiken erzeugt werden. Dieser Datenbestand besteht aus sogenannten Round-Robin-Archives. Dabei handelt es sich um Ableitungen aus den Primärdateien, die mit Hilfe statistischer Auswertungen ermittelt und für die festgelegten Zeitintervalle komprimiert werden. Die gesamte Anzahl der gespeicherten Datenwerte steht bereits bei der Erzeugung der RRD fest. Da auch zu Beginn alle Werte mit Platzhaltern aufgefüllt werden, bleibt die Größe einer RRD über die Zeit gleich, siehe Abbildung 6.

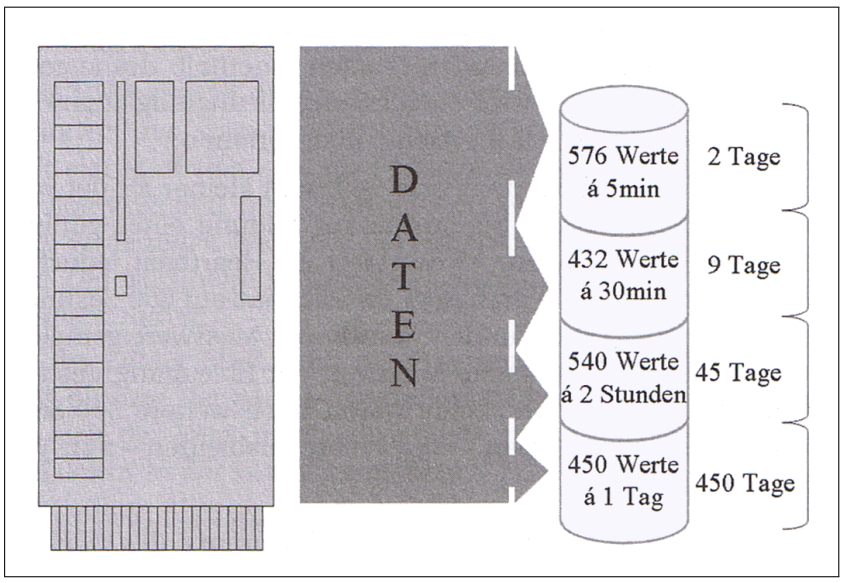

Abbildung 6: Modell der Round-Robin-Datenbanken[4]

[4]Quelle: [Munin08] S. 38

Munin verwendet dieses RRAs als hochauflösende Archive, die die Messwerte des aktuellen Tages, eine etwas komprimierte Fassung zur Darstellung der Daten der laufenden Woche sowie - noch stärker komprimiert - die Daten für den aktuellen Monat und das Jahr enthalten. Die Komprimierung wird so realisiert, dass Mittelwerte aus der feineren Zeitauflösung errechnet werden und zusätzlich noch spezielle Archivierungsregeln beim Erstellen der RRD-Datei berücksichtigt werden.

Aus diesen Datensätzen mit verschiedenen Zeitauflösungen erstellt Munin automatisch Graphen über die entsprechenden Zeiträume, siehe Abbildung 7.

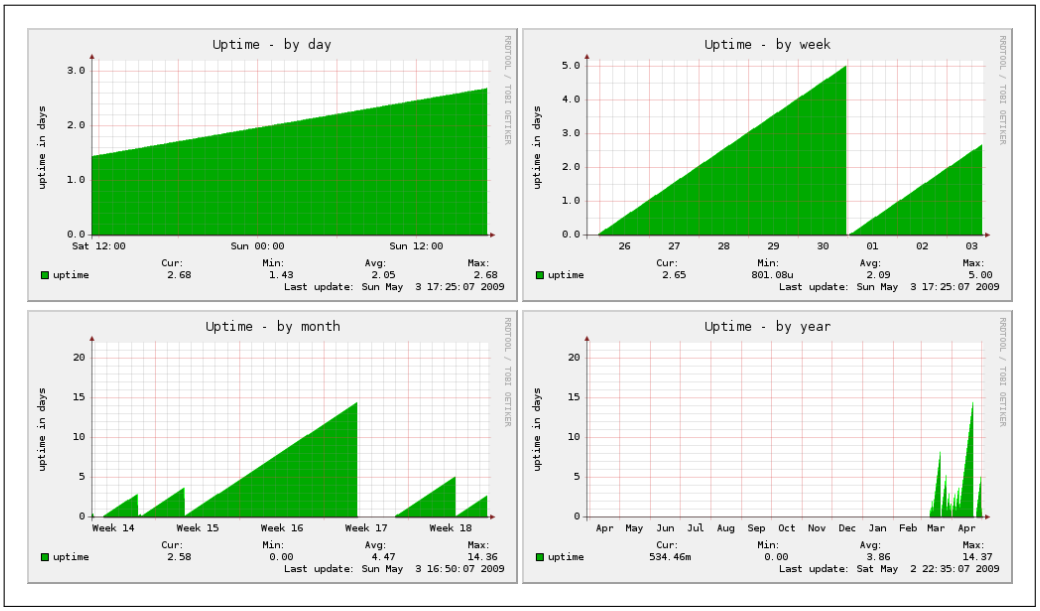

Abbildung 7: Übersicht der *Uptime*

3.3 Einbindung und Konfiguration der Plugins

Munin bringt bereits eine Sammlung von Plugins bei der Installation mit. Diese befinden sich in einem Bibliotheksverzeichnis. Wenn diese Plugins verwendet werden sollen um einen Node zu überwachen, müssen sie in einem Service-Verzeichnis verlinkt sein, damit der Daemon „`munin-node`“ darauf zugreifen kann. Dieser Daemon muss sich auf dem zu überwachenden Host befinden und dient als „Agent“ für den Munin-Master, da der Daemon auf seinem Port nach Aufforderungen durch den Master horcht. Ein solcher Daemon ist für Linux und für Windows verfügbar, wobei der für Windows weniger Funktionen bietet. Die einzelnen Tests laufen auf den Nodes unabhängig von der Abfrage des Masters. Wenn der Master die aktuellen Daten nicht abholt, wird dieser Datenbestand durch die neuen Informationen ausgetauscht und werden bei der nächsten Anfrage an den Master gesendet.
Auf den Graphen im Webinterface wird dieser Zeitraum als Lücke dargestellt, siehe Abbildung 8.

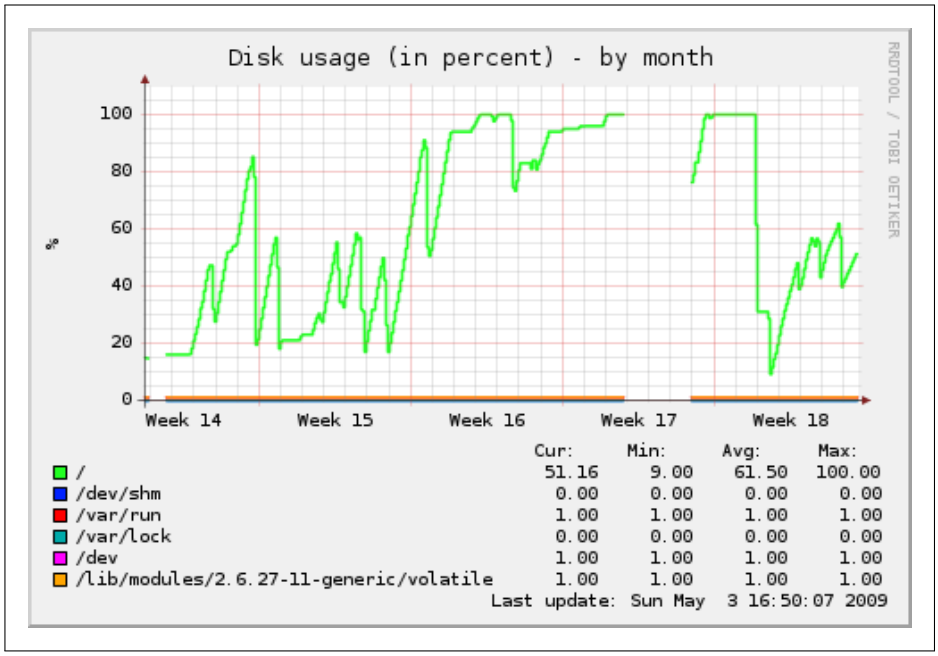

Abbildung 8: Fehlende Messwerte werden als Lücke dargestellt

Der „`munin-node`“-Daemon kannn so konfiguriert werden, dass er nur auf Anfragen von einer bestimmten IP reagiert, einen anderen Port verwendet oder nur auf einer bestimmten IP lauscht.
Damit der Daemon weiss, welche Daten er liefern muss, wird das zuvor erwähnte Service-Verzeichnis verwendet. Hier werden Verlinkungen zu den eigentlichen Plugins gespeichert. Eine beispielhafte Verlinkung wird in Abbildung 9 gezeigt.

```
paul@iwrpaul:~$ ln -s /opt/munin/lib/plugins/cpu /etc/opt/munin/plugins/cpu
```

Abbildung 9: Beispielhafte Verlinkung eines Munin-Plugins

3.3.1 Wildcard-Plugins

Falls der Name des Plugins mit einem Unterstrich endet, handelt es sich um ein sogenanntes *Wildcard*-Plugin. Solche Plugins erwarten in der Bezeichnung des Links spezifische Parameter für die Ausführung der Tests.
Beispielsweise verwendet das zuvor verwendete Plugin *ping_* als Parameter den Namen oder die IP-Adresse des anzupingenden Hosts. Die Verlinkung aus dem Bibliotheksverzeichnis in das Service-Verzeichnis würde dann folgendermaßen ausschauen:

```
paul@iwrpaul:~$ ln -s /opt/munin/lin/plugins/ping_ /etc/opt/munin/plugins/ping_localhost
```

Abbildung 10: Beispielhafte Verlinkung eines Wildcard-Plugins

Hier wird der Host *localhost* angepingt und diese Daten auf dem Master gespeichert und später in einem Graphen visualisiert.

3.3.2 SNMP-Plugins

Es ist auch möglich Informationen über den zu überwachenden Host in Erfahrung zu bringen, ohne, dass ein „`munin-node`“-Daemon auf dem Host installiert ist. Dies wird durch das Simple Network Management Protocol (SNMP) realisiert. Durch SNMP kann auf die strukturierte Datenhaltung der MIB in den entfernten Netzwerkknoten zugegriffen werden.

> „Die Management Information Base (MIB) dient als SNMP-Informationsstruktur und besteht aus einem hierarchischen, aus Zahlen aufgebauten Namensraum. Ähnliche Struktur wie andere hierarchische Verzeichnisdiensten wie DNS oder LDAP.“

Quelle: [Barth08] S.233

Die MIB-Struktur ist folgendermaßen aufgebaut:

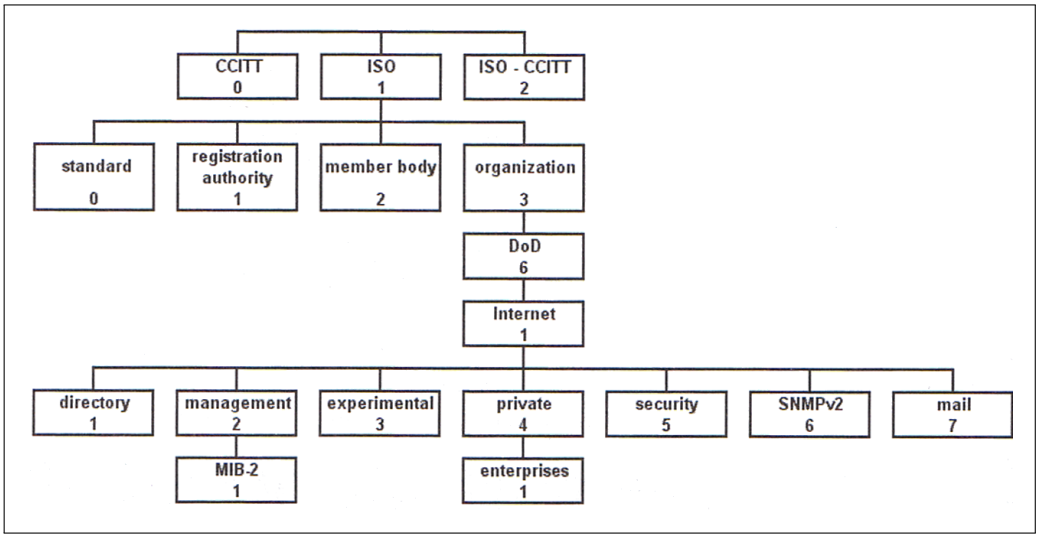

Abbildung 11: Struktur der Management Information Base (MIB)[5]

Dadurch können die SNMP-Plugins den gewünschten Wert über das Netzwerk abfragen, ohne, dass ein lokal auf dem Munin-Node installiertes Programm notwendig ist.

[5] Quelle: [Munin08] S. 156

Einen beispielhaften Zugriff auf SNMP-fähige Geräte wird in Abbildung 12 gezeigt.

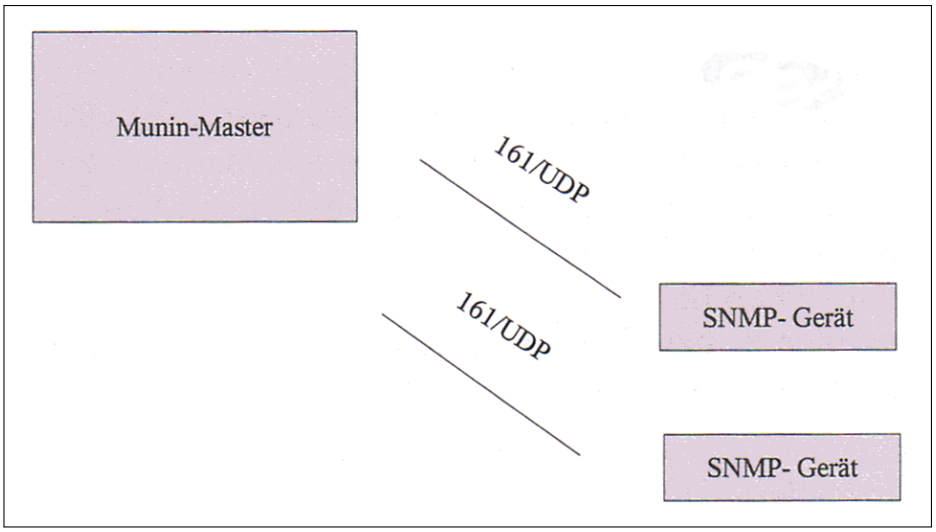

Abbildung 12: Beispielhafter Zugriff auf SNMP-fähige Geräte[6]

Munin verwendet für SNMP-Plugins eine besondere Namenskonvention. Alle Plugins dieser Art besitzen den Präfix *snmp_* auf den die IP-Adresse des SNMP-fähigen Gerätes folgt. Im folgenden Beispiel wird die Plattenbelegung eines entfernten Knotens überwacht.

```
paul@iwrpaul:~$ ln -s /opt/munin/lib/plugins/snmp__df /etc/opt/munin/plugins/snmp_192.168.2.1_df
```

Abbildung 13: Beispielhafte Verlinkung eines SNMP-Plugins

Auch bei diesen SNMP-Plugins gibt es Wildcard-Plugins. Dabei wird als Suffix im unteren Beispiel der zweite Netzwerkport eines SNMP-fähigen Switches nach Paketfehler abgefragt.

```
paul@iwrpaul:~$ ln -s /opt/munin/lib/plugins/snmp__if_err_ /etc/opt/munin/plugins/snmp_192.168.3.1_if_err_2
```

Abbildung 14: Beispielhafte Verlinkung eines Wildcard-SNMP-Plugins

[6]Quelle: [Munin08] S. 156

Munin liefert unter anderem folgende SNMP-Plugins mit:

- *snmp__df* überwacht die Plattenbelegung.
- *snmp__if_* ermittelt den Netzwerkdurchsatz.
- *snmp__if_err_* zählt die Paketfehler im Netzwerk.
- *snmp__sensors__fan* ermittelt die Lüfterdrehzahl.
- *snmp__load* überwacht die Systemlast.
- *snmp__processes* ermittelt die Anzahl der laufenden Prozesse.
- *snmp__users* ermittelt die Anzahl der eingeloggten Benutzer.

4 Umsetzung

Wenn die verwendete Linux-Distributon einen Paketmanager unterstützt verläuft die Installation der Anwendung einfach und ohne Probleme. Die hier als Testserver eingesetzte Ubuntu-Distribution verknüpft automatisch die Munin-Webseiten mit dem Webserver und verlinkt bereits die ersten verwendbare Munin-Plugins. Deshalb sind sofort nach der Installation einige Graphen verfügbar, siehe Abbildung 15.

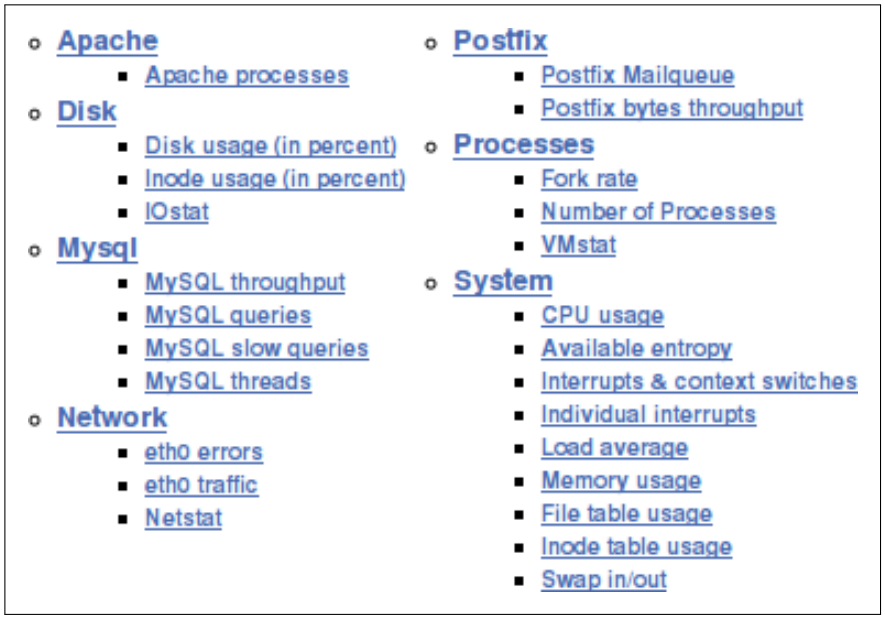

Abbildung 15: Bereits nach der Installation verfügbare Munin-Graphen

Dabei untersucht Munin das verwendete System während der Installation nach verfügbaren Überwachungssobjekten. Zum Beispiel wurde in der oberen Abbildung von Munin der Mailserver *postfix* gefunden und automatisch in eine eigene Kategorie in das Webinterface eingepflegt.
Wie bereits in Kapitel 3.3 beschrieben, können weitere mitgelieferte Munin-Plugins in das entsprechende Service-Verzeichnis verlinkt und dabei ggf. mit Parametern ausgestatten werden, damit Munin zusätzlich diese Graphen erstellt und deren Werte aktualisiert und abspeichert.

Dabei gibt es noch eine zusätzliche Hilfe durch die Plugins, sofern sie sich an den Munin-Plugin-Standard halten. Solche Skripte überprüfen selbständig bei ihrer Ausführung aufgrund welcher Bedingung sie nicht genutzt werden können oder sogar mit welchen Parametern sie funktionieren würden.
Diese hilfreiche Unterstützung bekommt man durch die Ausführung des folgenden Befehls:

```
paul@iwrpaul:~$ munin-node-configure --suggest
Plugin                  | Used | Suggestions
------                  | ---- | -----------
acpi                    | no   | [thermal not supported by ACPI]
apache_accesses         | no   | [no apache server-status or ExtendedStatus missing on ports 80]
apache_volume           | no   | [no apache server-status or ExtendedStatus missing on ports 80]
courier_mta_mailqueue   | no   | [spooldir not found]
courier_mta_mailstats   | no   | [could not find executable]
courier_mta_mailvolume  | no   | [could not find executable]
cupsys_pages            | no   | [logfile not found]
exim_mailqueue          | no   |
exim_mailstats          | no   | ['/usr/sbin/exim -bP log_file_path' returned an error]
hddtemp_smartctl        | no   | [no drives known]
if_                     | yes  |
if_err_                 | yes  |
ip_                     | no   |
mysql_isam_space_       | no   |
nfs_client              | no   | [no /proc/net/rpc/nfs]
nfsd                    | no   | [no /proc/net/rpc/nfsd]
ntp_offset              | no   | yes
ps_                     | no   |
sendmail_mailqueue      | no   |
sendmail_mailstats      | no   | [no mailstats command]
sendmail_mailtraffic    | no   |
smart_                  | no   |
squid_cache             | no   | [could not connect: Connection refused]
squid_requests          | no   | [could not connect: Connection refused]
squid_traffic           | no   | [could not connect: Connection refused]
tomcat_access           | no   | [no tomcat status]
tomcat_jvm              | no   | [no tomcat status]
tomcat_threads          | no   | [no tomcat status]
tomcat_volume           | no   | [no tomcat status]
```

Abbildung 16: Automatische Überprüfung der Munin-Plugins

Anhand dieser Informationen kann man herausfinden, weshalb ein Plugin nicht richtig funktioniert oder nicht automatisch bei der Installation hinzugefügt wurde.

4.1 Erstellen eines eigenen Munin-Plugins

Die zum Zeitpunkt dieser Arbeit mitgelieferten 132 Munin-Plugins unterstützen eine Vielzahl an verschiedenen Überwachungsmöglichkeiten. Jedoch kann es auch sein, dass ein gewünschtes Element oder Detail nicht von den mitgelieferten Plugins beachtet wird. Dann bietet es sich an auf der MuninExchange[7] Internetseite nach bereits entwickelten passenden Munin-Plugins zu stöbern.

Eine weitere Möglichkeit besteht darin das gewünschte Skript selbst zu entwickeln. Da der Aufbau eines Munin-Plugins recht primitiv und klein ausfallen kann, benötigt man keine umfangreiche Programmierkenntnisse um erfolgreich ein funktionierendes Skript zu erstellen.

Im folgenden soll als Beispiel eine einfache Speicherplatzüberwachung eines Ordners mit Munin realisiert werden.

Dafür wird das bereits vorhandene Systemtool *du* verwendet. Den Speicherplatzbedarf eines Ordners lässt sich dann folgendermaßen ermitteln:

```
paul@iwrpaul:~$ du /test
449731  /test
paul@iwrpaul:~$ du -sm /test
440     /test
paul@iwrpaul:~$ du -sm /test | cut -f1
440
```

Abbildung 17: Ermittlung des Speicherplatzbedarf des Ordners */test*

Mit dem ersten Befehl wird die Verzeichnisgröße des Ordners */test* gemessen. Der dabei ermittelte Rückgabewert besitzt KByte als Einheit; da es sich aber um einen etwas größeren Ordner handelt und damit der spätere Munin-Graph nicht zu unübersichtlich wird, bietet es sich an diese Einheit mit dem zweiten Befehl durch den zusätzlichen Parameter *-sm* in MByte umzurechnen. Da

[7] http://muninexchange.projects.linpro.no/

nur der Speicherplatzbedarf des Ordners interessiert, wird der zweite Teil der Ausgabe einfach mit dem *cut*-Befehl abgeschnitten.

Eine für Munin verwertbare Ausgabe, wie in Abbildung 2, muss einen internen Variablennamen, in diesem Fall *dir* gefolgt von dem Suffix *.value* besitzen. Daraus ergibt sich folgendes kurzes BASH-Skript:

```
1  #!/bin/bash
2  # Plugin to monitor the size of the specified directory
3  #
4  # directory to check
5  DIR="/test"
6  ######################################################################
7
8  echo -n "dir.value "
9  if [ -d $DIR ]; then #check if the dir exists
10     SIZE=‘du -sm $DIR | cut -f1‘
11     echo $SIZE
12     exit 0
13 else
14     echo "Error: check your directory!"
15     exit 1
16 fi
```

Listing 2: Speicherplatzbedarf eines Verzeichnisses

Nach der Verlinkung in das entsprechende Service-Verzeichnis gibt das Aufühen des Skriptes folgende Ausgabe:

```
root@iwrpaul:~# ln -s /usr/share/munin/plugins/du /etc/munin/plugins/
root@iwrpaul:~# /etc/munin/plugins/du
dir.value 440
```

Abbildung 18: Ausführung des Testplugins

Mit dieser Ausgabe kann „`munin-update`“ die Messwerte ermitteln und speichern.

Für die Visualisierung dieser Messwerte benötigt Munin jedoch zusätzlich noch Informationen, wie der Graph auszusehen hat. Für diese Informationen

ruft Munin die Plugins mit dem Parameter *config* auf; deshalb muss auch dieses Plugins die entsprechende Werte bei diesem Parameteraufruf liefern. Folgende Informationen werden für das Testplugin benötigt:

- *graph_title* als Titel für den Graphen
- *graph_vlabel* als Beschriftung der y-Achse
- *graph_category* für die Kategorie in welche der resultierende Graph eingeordnet werden soll
- *dir.label* als Bezeichnung der Messwerte
- *dir.min* als Minimalwert der Messwerte
- *dir.info* zusätzliche Beschreibung unterhalb des Graphen

Der dafür benötigte Quelltext wird vor der eigentlichen Messwertermittlung eingeschoben, da sich das Skript ansonsten zuvor mit *exit* beendet. Das fertige Skript sieht dann folgendermaßen aus:

```
1  #!/bin/bash
2  # Plugin to monitor the size of the specified directory
3  #
4  # directory to check
5  DIR="/test"
6  ######################################################################
7
8  if [ "$1" = "config" ]; then
9          echo "graph_title Directory size: $DIR"
10         echo "graph_vlabel size MB"
11         echo "graph_category disk"
12         echo "graph_info Size of $DIR"
13         echo "dir.label size"
14         echo "dir.min 0"
15         echo "dir.info Shows du -sm for specified directory"
16         exit 0
17 fi
18
```

```
19 echo -n "dir.value "
20 if [ -d $DIR ]; then #check if the dir exists
21     SIZE=‘du -sm $DIR | cut -f1‘
22     echo $SIZE
23     exit 0
24 else
25     echo "Error: check your directory!"
26     exit 1
27 fi
```

Listing 3: Fertiges Skript für den Speicherplatzbedarf eines Verzeichnisses

Nach dem Neustarten des „`munin-node`“-Daemons wird dann folgender Graph im Webinterface angezeigt:

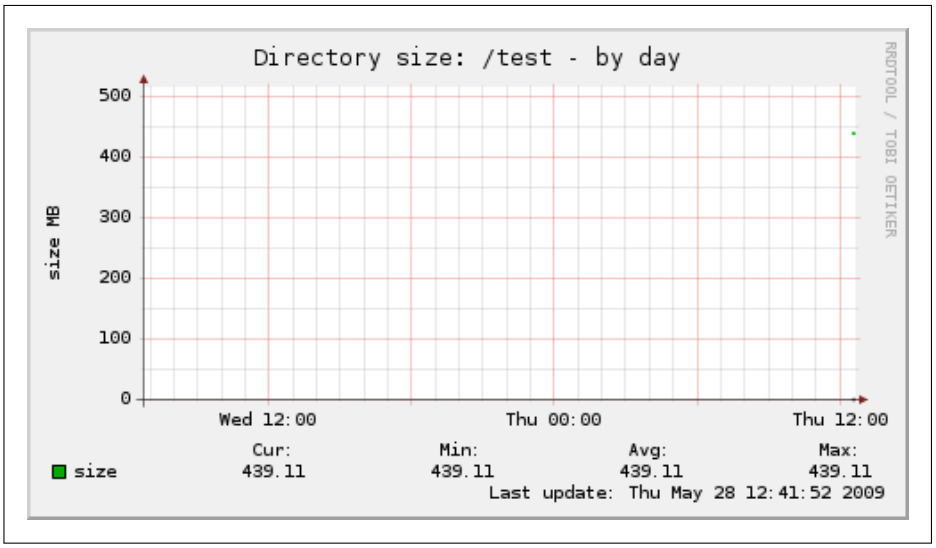

Abbildung 19: Fertiger Muningraph des Testskriptes

Dies ist nur ein simples Beispiel für ein eigenes Munin-Plugin. Dem Entwickler sind aber fast keine Grenzen gesetzt, solange er sich an die von Munin benötigten Werte hält.

4.2 Zusammenspiel mit Nagios

Nagios dient zum Überwachen von Hosts und deren Services in komplexeren Infrastrukturen und bietet im Gegensatz zu Munin viele zusätzliche Features. Es gibt einige essentielle Unterschiede zwischen den beiden Überwachungsanwendungen. Während Munin mehr Wert auf die Visualisierung der Überwachungsdaten legt, kümmert sich Nagios mehr um die Überwachungs- und Alarmierungslogik. Beispielsweise überwacht Munin ständig die Überschreitung von den angegebenen Schwellwerten und schlägt ggf. Alarm. Im Gegensatz hierzu wird bei Nagios mit Hard- und Soft-States gearbeitet, bei denen sich der Fehler erst durch mehrmaliges Überprüfen als „True Positive“ beweisen muss.

Dies ist nur ein Beispiel für die überlegene Überwachungslogik von Nagios, doch Munin wurde nicht in der Absicht entwickelt mit Nagios zu konkurrieren. Der genaue Gegenteil ist der Fall - Nagios und Munin können zusammenarbeiten.

Es ist auch durchaus denkbar, dass bereits ein Nagios-Überwachungssytem in der Netzwerkstruktur betrieben wird. Eine zusätzliche Benachrichtigung von Fehlern bzw. Warnungen und die dezentrale Aufsplittung der Überwachung in zwei getrennte Systeme ist in den meisten Fällen unlogisch und nicht gewollt. Deshalb gibt es die Möglichkeit die vorhandene Nagios-Instanz als „Event-Handler“ für die von Munin festgestellten Überschreitungen von Schwellwerten zu benutzen.

Dabei wird anstatt der Alarmierung durch Emails ein passives Check-Ergebnis mittels „`send_nsca`“ an den Nagios Server versendet. Dieser nimmt das Ergebnis entgegen und benachrichtigt die entsprechenden Benutzer. Damit Nagios solche passiven Ergebnisse aufnehmen kann, muss sich auf dem Nagios-Server ein NSCA-Daemon (Nagios Service Check Acceptor) befinden, der auf dem entsprechenden Port auf die Benachrichtigungen wartet.

In folgender Abbildung wird ein beispielhafter Aufbau eines solchen Systems gezeigt, welches die Kommunikation zwischen Nagios und Munin verdeutlichen soll:

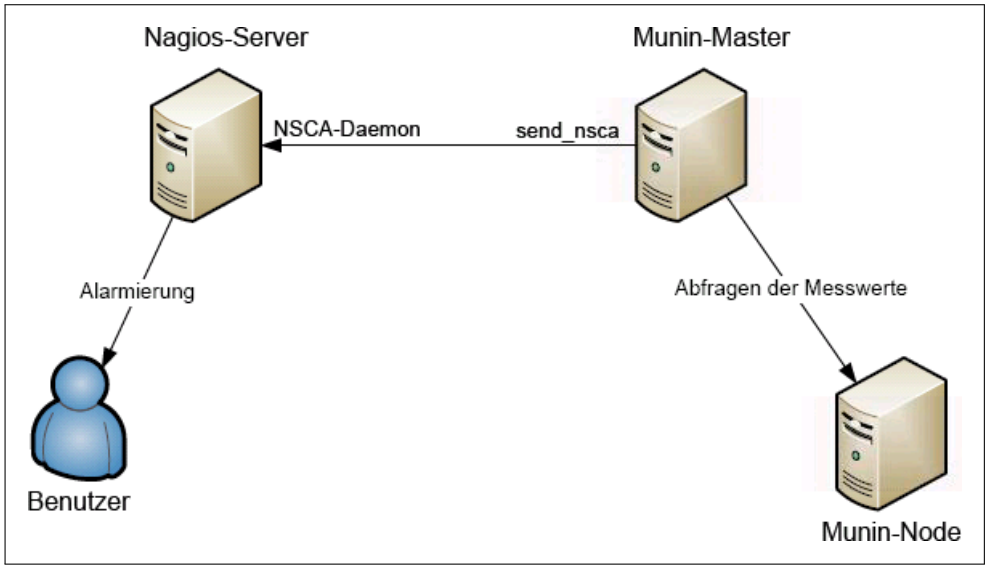

Abbildung 20: Zusammenarbeit von Munin und Nagios

1. Der Munin-Master sammelt in periodischen Abläufen die Messwerte ab, welche ständig von den Munin-Nodes selbst ermittelt werden.

2. Die Messwerte werden vom Master überprüft.

3. Bei einer Überschreitung wird anstatt einer Email das Tool „`send_nsca`“ dazu verwendet, den Nagios-Server darüber zu benachrichtigen.

4. Der auf dem Nagios-Server laufende NSCA-Daemon nimmt das Ergebnis des Munin-Masters als passives Check-Ergebnis entgegen und filtert nochmals nach seinen eigenen Benachrichtigungs- und Eskalationsregeln.

5. Bei einer notwendigen Alarmierung nimmt Nagios Kontakt mit den zuständigen Benutzern auf.

4.3 Vergleich der Visualisierung mit Nagios

Nagios bietet standardmäßig keine Visualisierung der Überwachungsdaten an.

Durch verschiedene Addons lässt sich dies jedoch nachträglich hinzufügen.

Solche Addons sind zum Beispiel:

- Nagiosgraph

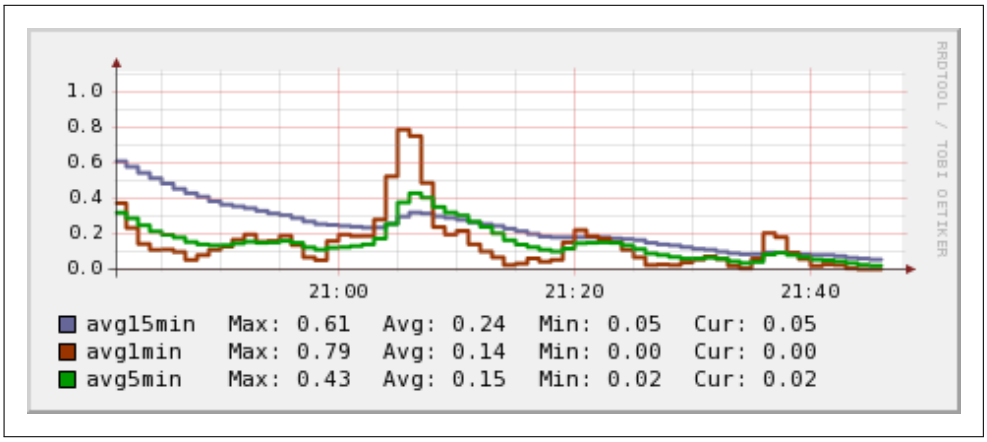

Abbildung 21: Visualisierung der Performancedaten mit Nagiosgraph

- NagiosGrapher

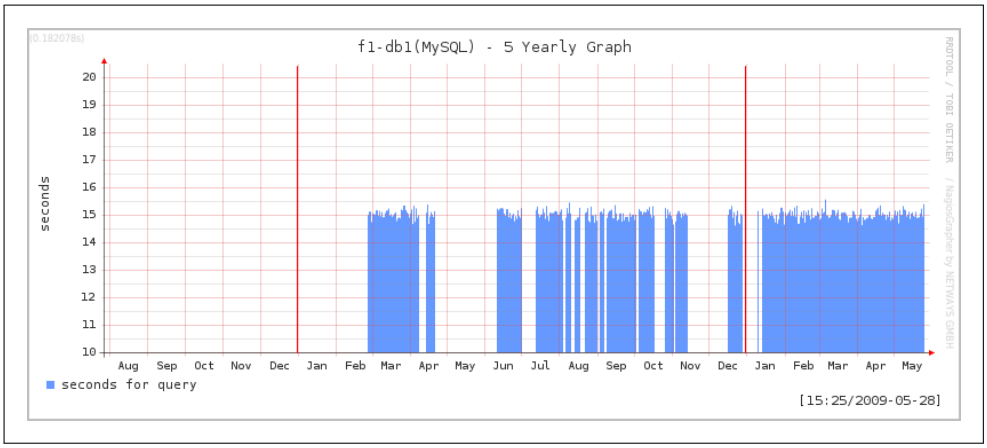

Abbildung 22: Visualisierung der Performancedaten mit NagiosGrapher

- drraw

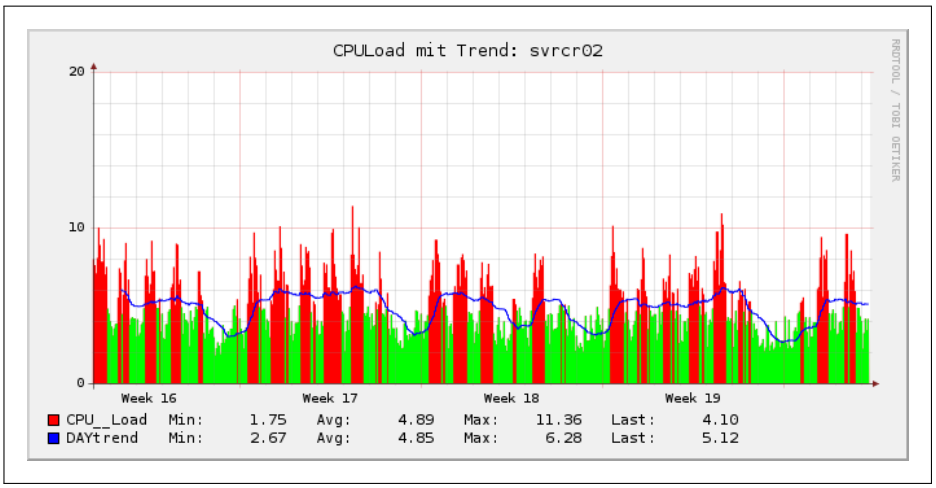

Abbildung 23: Visualisierung der Performancedaten mit drraw

- Perf2rrd

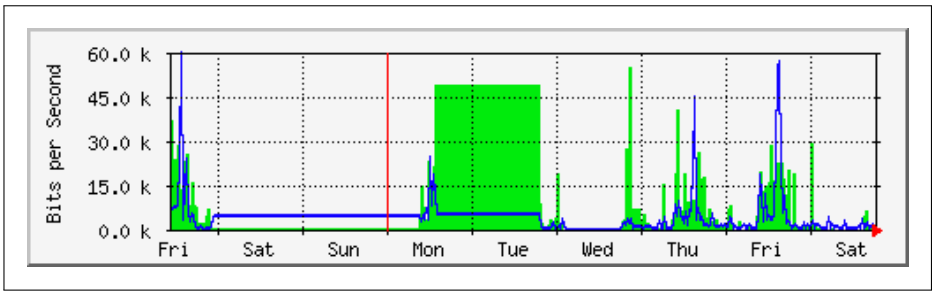

Abbildung 24: Visualisierung der Performancedaten mit Perf2rrd

Allgemein lässt sich sagen, dass die Visualisierung der Überwachungsdaten durch Nagios als unerfahrener Benutzer deutlich aufwändiger ist als mit Munin. Da alleine die Entscheidung für ein Addon und dessen Basiskonfiguration bis zum ersten Graphen länger als eine vollständige Munin Installation dauert. Dafür stehen mit den Nagios-Visualisierungs-Addon mehr Möglichkeiten zur Verfügung, indem zum Beispiel die Performanzedaten länger als ein Jahr gespeichert werden können, wie in vorheriger Abbildung 22 mit einem Zeitverlauf über fünf Jahre gezeigt wird.

5 Zusammenfassung und Ausblick

Die von Munin zusammengetragenen Informationen sind dank der einfachen Installation der Anwendung und der gewünschten Munin-Plugins sehr schnell in nützliche Graphen umgewandelt und erweisen sich dank der verschiedenen Zeitauflösung als sehr nützliches Element in der Langzeitüberwachung.
Auch die große Anzahl an bereits bei der Installation verfügbaren Plugins lassen sich dank einer automatischen Testfunktion, siehe Abbildung 16 in Kapitel 4, schnell und unkompliziert einbinden.
Sollte sich doch nicht das passende Plugin in dieser Sammlung finden, gibt es immernoch die Möglichkeit auf der MuninExchange Internetseite fündig zu werden. Dort befinden sich von anderen Munin-Benutzern entwickelte und veröffentlichte Plugins, die auch weiter vom Benutzer auf die eigenen Bedürfnisse angepasst werden können. Da die Plugins jedoch recht einfach aufgebaut sind, lassen sich auch schnell und einfach eigene Kreationen entwerfen, die sofort automatisch nach der Verlinkung zum Service-Verzeichnis in das Webinterface eingefügt werden.
Munin legt großen Wert auf die Visualisierung der Überwachungsinformationen und weniger auf eine komplexe, umfangreiche Überwachungs- und Alarmierungslogik. Dafür empfiehlt es sich für diesen Zweck Nagios zu verwenden und Munin in das Nagios Überwachungssytem einzubinden. Denn sowohl Nagios als auch Munin bieten die notwendigen Werkzeuge um dies zu realisieren. Dabei muss erwähnt werden, dass auch mit Nagios die Visualisierung der Performancedaten der Plugins möglich ist, so dass Munin als überflüssig gesehn werden könnte. Jedoch stellt sich die Konfiguration und Anpassung von Nagios und dem Visualisierung-Addon für den unerfahrenen Benutzer als deutlich schwieriger und aufwändiger heraus als der Aufbau eines ähnlichen Munin Überwachungssystems.

6 Abbildungsverzeichnis

7 Tabellenverzeichnis

8 Codelistingverzeichnis

9 Literaturverzeichnis

[Pro02] Ernst Probst (2002) „Nordische Göttersagen“,
ISBN13: 978-3-936326-05-5, Einsichtnahme: 02.05.2009

[Munin08] Gabriele Pohl und Michael Renner (2008) „Munin - Graphisches Netzwerk- und System-Monitoring“,
ISBN13: 978-3-937514-48-2, Einsichtnahme: 05.04.2009

[Barth08] Wolfgang Barth (2008) „Nagios - System- und Netzwerk-Monitoring“ 2. Auflage,
ISBN13: 978-3-937514-46-8, Einsichtnahme: 14.05.2009